
Aplicación multiplataforma para transformación personalizada de textos



Jaime Delgado Linares
Juan Luis García Flores

Directores:
Raquel Hervás Ballesteros
Susana Bautista Blasco

Trabajo de fin de grado del Grado en Ingeniería informática
Facultad de Informática
Universidad Complutense de Madrid 2015

Aplicación multiplataforma para transformación personalizada de textos

**Jaime Delgado Linares
Juan Luis García Flores**

Directores:
**Raquel Hervás Ballesteros
Susana Bautista Blasco**

**Trabajo de fin de grado del Grado en Ingeniería informática
Facultad de Informática
Universidad Complutense de Madrid 2015**

Agradecimientos

Para empezar, queremos dar las gracias a Raquel y Susi por todo el tiempo que nos han dedicado, porque ha sido todo una experiencia trabajar con ellas, y porque, ya que se han leído esta memoria tantas y tantas veces, no podían faltar en los agradecimientos.

También, gracias a Pablo y Guillermo por hacernos las evaluaciones heurísticas al final del curso, cuando menos tiempo tenían, y aún así nos hicieron un hueco. Las evaluaciones han sido imprescindibles en este proyecto.

Y para finalizar, gracias a todos los familiares, amigos y profesores que nos han ayudado todo este tiempo.

Autorización

Se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria como el código, la documentación y/o el prototipo desarrollado.

Jaime Delgado Linares

Juan Luis García Flores

Resumen

Cualquier persona puede tener problemas a la hora de comprender un texto. Algunas técnicas para comprender mejor los textos se suelen basar en subrayar aquellas partes que resulten más importantes, extraer las palabras que no se entienden para su posterior estudio o incluso anotar en el propio texto algunas notas que ayuden a comprender mejor ciertas partes del texto. Por otro lado, vivimos en la sociedad de la información en la que el uso de dispositivos como ordenadores, *tablets* y *smartphones* está cada vez más extendido. En este contexto, el uso de internet a través de estos dispositivos se ha convertido en una herramienta prácticamente imprescindible en el ámbito laboral, cultural, social y del ocio.

Este proyecto pretende ayudar a estas personas que necesitan comprender mejor textos facilitando la transformación o enriquecimiento personalizado del texto para facilitar su comprensión. Existen varios sistemas que llevan a cabo esta labor de diferentes formas, desde aplicaciones que ayudan a personas invidentes, aplicaciones específicas para ayudar a niños pequeños o a gente mayor, aplicaciones que pretenden ayudar a aprender idiomas o incluso a facilitar la navegación por internet. En nuestro proyecto desarrollaremos un sistema web para facilitar a los usuarios la comprensión de los textos que deseen poniendo a su disposición un conjunto de herramientas para tal fin que se basarán en enriquecer el texto con las acciones de subrayar, definir, traducir a inglés, extraer sinónimos y antónimos y analizar morfológicamente ciertas palabras.

El proyecto se divide en dos partes. La primera es el desarrollo de un servicio web que hace uso de otros servicios web útiles para el enriquecimiento de texto. Tanto este servicio web como los servicios web que usa, están a disposición de cualquier persona pudiendo hacer uso del mismo de forma libre. La segunda parte consiste en el desarrollo de una aplicación web multiplataforma que haga uso del servicio web mediante la cual el usuario sea capaz de enriquecer el texto según sus necesidades utilizando una serie de funciones. Ambas partes han sido desarrolladas para que sean fáciles de utilizar por cualquier persona, sin necesidad de amplios conocimientos informáticos.

Abstract

Anyone can have problems understanding a text. Some techniques to understand better the texts are often based on emphasize those parts that are more important, extract the words that are not understood for further study or even write down in the text some notes to help you to understand better certain parts of the text. On the other side, we live in the information society where the use of dispositives as computers, tablets and smartphones is growing extended. In this context, the use of Internet through these devices has become an almost indispensable tool in the labor, cultural, social field and leisure.

This project aims to help these people who need to understand better texts facilitating transformation or personal enrichment of the text to help their understanding. There are several systems that lead out this task in different ways, from applications that help blind people, specific applications to help young children or older people, applications that claim to help learning languages or even to facilitate internet browsing. In our project we will develop a web system to make it easier for users to understand the texts wished by providing a set of tools for this purpose that enrich the text based on the actions of underline, define, translate into English, synonyms and antonyms extract and analyze morphologically certain words.

The project is divided into two parts. The first is the development of a web service that use others useful web services for the enrichment of the text. Both, the web service and web services used are available so anyone can make use of it freely. The second part is the development of a web multiplatform application that makes use of web service through which the user is able to enrich the text according to their needs using a variety of functions. Both sides have been developed to be easy to use by anyone without extensive computing knowledge.

Palabras Clave

- Enriquecimiento de texto
- Procesamiento de Lenguaje Natural
- Diseño Guiado por Objetivos
- Multiplataforma
- Personalización de texto
- Accesibilidad

Keywords

- Text enrichment
- Natural Language Processing
- Goal-Centered Design
- Multi-platform
- Text personalization
- Accessibility

Índice

Agradecimientos	V
Autorización	VII
Resumen	IX
Abstract	XI
Palabras Clave	XIII
Keywords	XV
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura del documento	2
2. Introduction	5
2.1. Motivation	5
2.2. Objectives	6
2.3. Document structure	6
3. Estado del arte	9
3.1. Procesamiento del Lenguaje Natural (PLN)	9
3.1.1. Análisis del lenguaje	9
3.1.2. Analizadores lingüísticos	10
3.1.3. Herramientas de análisis a través de servicios web . . .	11
3.2. Tecnologías de servicios web	13
3.2.1. Protocolos de servicios web	14
3.2.2. Transferencia de Estado Representacional (REST) . .	16
3.2.3. Servicios web RESTful	17
3.2.4. JAX-RS	19
3.2.5. JAX-WS	19

3.3. Interfaces de usuario con HTML5	20
3.3.1. HTML5	20
3.3.2. CSS3	21
3.3.3. JavaScript	22
3.3.4. JQuery y JQuery Mobile	23
3.3.5. Bootstrap	23
3.3.6. Material design	23
3.3.7. JSON	24
3.4. Trabajos relacionados	25
3.4.1. PorSimples	25
3.4.2. NavegaFácil	26
3.4.3. Open Book	28
4. Diseño y requisitos	33
4.1. Diseño Guiado por Objetivos (DGO)	33
4.1.1. Fases del Diseño Guiado por Objetivos	34
4.2. Adaptación del Diseño Guiado por Objetivos para el proyecto	42
4.3. Desarrollo del diseño en el proyecto	44
4.3.1. Elementos de datos y elementos funcionales	44
4.3.2. Boceto del framework de interacción	46
4.3.3. Construcción de los escenarios key path	46
4.3.4. Validación del prototipo de baja fidelidad con usuarios	49
4.3.5. Informe de hallazgos y recomendaciones	50
5. Arquitectura	53
5.1. Aplicación web	54
5.2. Servidor	55
6. Aplicación para el enriquecimiento personalizado de textos	57
6.1. Uso de la aplicación y elementos que la componen	58
6.1.1. CKEditor	59
6.1.2. Enriquecer texto	60
6.1.3. Panel de piezas y acciones	61
6.1.4. Cajas de piezas y acciones	64
6.1.5. Botón aplicar	65
6.1.6. Discusión sobre las evaluaciones iniciales	69
7. Servidor	71
7.1. Servicio principal	71
7.2. Llamada al servicio principal	74
7.3. Funcionamiento del servicio principal	74
7.4. Etapas del servicio principal	75

7.4.1. Inicializar ejecución	75
7.4.2. Eliminar enriquecido	76
7.4.3. Tokenización del texto	77
7.4.4. Unir acciones y tokens	81
7.4.5. Aplicar la ejecución	81
7.4.6. Guardar y mostrar	83
8. Evaluaciones heurísticas	87
9. Conclusiones y trabajo futuro	89
9.1. Conclusiones	89
9.2. Trabajo futuro	90
10. Conclusions and future work	93
10.1. Conclusions	93
10.2. Future work	94
11. Trabajo individual	97
11.1. Jaime Delgado Linares	97
11.2. Juan Luis García Flores	99
Bibliografía	101

Índice de figuras

3.1. Ejemplo de análisis sintáctico.	10
3.2. Ejemplo de salida Maltparser.	12
3.3. Árbol de dependencias de Maltparser.	13
3.4. Ejemplo de salida Freeling.	13
3.5. Estructura de la pila de protocolos para servicios web.	15
3.6. Arquitectura REST.	17
3.7. Ejemplo de petición GET al servicio web de Youtube.	18
3.8. Ejemplo de página web simple en HTML.	20
3.9. Ejemplo simple de árbol DOM.	22
3.10. Interfaz de Whatsapp con Material design.	24
3.11. Ejemplo de una persona en formato JSON.	25
3.12. Sistema FACILITA de PorSimples.	27
3.13. Sistema SIMPLIFICA de PorSimples.	27
3.14. Sistema FACILITA Educacional de PorSimples.	28
3.15. Funciones que ofrece NavegaFácil.	29
3.16. Configuración de NavegaFácil.	30
3.17. Editor Open book.	31
4.1. Ejemplo de persona.	35
4.2. Ejemplo de prototipo en papel interactivo.	39
4.3. Pantalla inicial del sistema en el boceto.	47
4.4. Pantalla para aplicar piezas y acciones	48
4.5. Panel lateral izquierdo con más opciones.	49
5.1. Arquitectura y comunicación del sistema	53
5.2. Arquitectura de la aplicación web	54
5.3. Arquitectura del servicio web	55
6.1. Página principal de la aplicación web.	58
6.2. Versión estandar de CKEditor.	59
6.3. Página de enriquecimiento.	61
6.4. Pestaña Piezas.	62

6.5. Pestaña Acciones.	62
6.6. Resultado de aplicar la acción Subrayar.	63
6.7. Resultado de aplicar la acción Análisis Morfológico.	64
6.8. Resultado de aplicar la acción Sinónimos.	65
6.9. Resultado de aplicar la acción Antónimos.	66
6.10. Resultado de aplicar la acción Definiciones.	66
6.11. Resultado de aplicar la acción Traducción a inglés.	67
6.12. Resultado de aplicar las acciones Subrayar y Análisis Morfológico.	67
6.13. Resultado de aplicar dos filas.	68
6.14. Caja de acciones vacía.	68
6.15. Diálogo modal.	69
7.1. Código JavaScript de la petición AJAX	75
7.2. Versión web de Freeling	80
7.3. Ejemplo de salida de Freeling	81
7.4. Ejemplo de resultados de wordreference para sinónimos y antónimos	83
7.5. Ejemplo de resultados de wordreference para definiciones	84
7.6. Ejemplo de resultados de WordReference para traducciones	85

Índice de Tablas

1.1. Resultado	1
2.1. Result	5
3.1. Categorías gramaticales de las palabras	10
4.1. 10 Criterios de Nielsen	41
4.2. 8 Reglas de oro de Shneiderman	41
6.1. Piezas ofrecidas por el sistema	61
7.1. Formato de las piezas en el objeto JSON	73
7.2. Formato de las piezas en el objeto JSON	73
8.1. Puntuaciones asignadas a las tareas.	88

Capítulo 1

Introducción

1.1. Motivación

Muchas veces nos encontramos con que tenemos que trabajar con textos complejos y nos gustaría que estos textos fueran mucho más comprensibles. A lo mejor simplemente con sacar unas palabras claves nos serviría para acortar mucho el trabajo. Al final, nos encontramos con que dedicamos demasiado tiempo a comprender el texto. Entonces, sería interesante construir una aplicación en la que se pudiera extraer fácilmente ciertos fragmentos del texto, analizarlos a nuestro gusto y añadir esa información al texto para poder comprenderlo con mayor facilidad.

Por ejemplo, si estamos estudiando unos apuntes de Historia, sería muy interesante sacar los nombres del texto o incluso sacar las fechas. De este modo tendríamos una visión general del texto y podríamos hacer un resumen por extracción. Por ejemplo:

“Cristobal Colón es famoso por haber descubierto América, el 12 de octubre de 1492, al llegar a la isla de Guanahani, actualmente en Las Bahamas..”

En la Tabla 1.1 podemos ver los nombres y las fechas de este texto que obtendríamos:

Cristobal Colón	América	12 de octubre de 1492
isla	Guanahani	Las Bahamas

Tabla 1.1: Resultado

Viendo estos resultados, nos podemos hacer un poco la idea de qué va el texto y podría ayudar mucho al proceso de simplificación.

Otro ejemplo de uso interesante, sería alguien que quiera aprender vocabulario en inglés. Sería muy útil, por ejemplo, poder sacar del texto todos los adjetivos o todos los nombres de un texto, y saber rápidamente todas las tra-

ducciones. Además, la persona que esté aprendiendo inglés tendría todo ese enriquecimiento disponible en su texto, y podría estudiarlo más fácilmente.

Del mismo modo, también serviría para ayudar a aprender vocabulario del castellano, como pueden ser sinónimos o antónimos de palabras, definiciones, o incluso un análisis morfológico de las palabras del texto.

Nuestro objetivo en este proyecto será hacer una aplicación multiplataforma que se pueda usar por cualquier persona, que permita hacer transformaciones personalizadas en textos que ayuden a enriquecer el texto para facilitar su entendimiento. Además añadiremos opciones adicionales como pueden ser la de análisis morfosintáctico, traducción al inglés, sinónimos, antónimos e incluso definiciones, que ofrecerán muchas posibilidades.

Ya hay aplicaciones que hacen labores similares, pero nuestro trabajo será unir las todas en una aplicación que sea fácil de utilizar, que tenga una apariencia agradable para el usuario y que se pueda ejecutar en distintos dispositivos.

1.2. Objetivos

En vista de las necesidades que tienen las personas para la comprensión de texto escrito se enumeran los siguientes objetivos a cumplir:

- Investigación de otros sistemas que tengan la misma motivación o similar a la de este proyecto.
- Investigación de tecnologías disponibles y elección de las más adecuadas e investigación de servicios web útiles para enriquecer texto.
- Diseño de una arquitectura eficiente del sistema que combine las tecnologías elegidas.
- Diseño de una interfaz usable y validada por usuarios y expertos en este campo de la usabilidad.
- Creación de un servicio web que comunique con otros servicios web que ayuden al enriquecimiento del texto. Construcción de un cliente que comunique con el servicio web.
- Publicación del sistema en un repositorio o en la web.

1.3. Estructura del documento

El capítulo 3 hace un repaso de las tecnologías que se utilizarán en el trabajo, así como otros sistemas existentes en el campo de la adaptación de

texto para facilitar su comprensión. Se hablará de procesamiento de lenguaje natural, tecnologías referentes a aplicaciones web y servicios web, y de sistemas relacionados con el trabajo presentado.

En el capítulo 4 se cuenta el proceso llevado a cabo para definir la interfaz del sistema, así como el comportamiento de la aplicación y la forma de interactuar con ella. Se contará qué es el Diseño Guiado por Objetivos, cómo se ha adaptado para este proyecto y el desarrollo de un primer prototipo en papel y la validación del mismo con usuarios.

El capítulo 5 explica la arquitectura del sistema, primero con un esquema general de la arquitectura y después introduciendo cada uno de los componentes que se desarrollarán en capítulos posteriores con mayor detalle.

El capítulo 6 cuenta el uso de la aplicación y los elementos que la componen junto con la función que desempeñan.

En el capítulo 7 se detalla cómo está implementada la parte del servidor. Se contará el servicio principal que se comunica con la aplicación, su funcionamiento, la llamada a este servicio y las etapas que se llevan a cabo para la modificación o enriquecimiento del texto.

El capítulo 8 presenta las evaluaciones heurísticas que se hicieron con expertos de la usabilidad.

En los capítulos 9 y 10 se exponen las conclusiones a las que se han llegado al desarrollar el proyecto y las posibles mejoras o modificaciones que podrían añadirse en futuras iteraciones.

Por último, en el capítulo 11 se exponen las aportaciones de ambos miembros del grupo durante el desarrollo del proyecto.

Capítulo 2

Introduction

2.1. Motivation

Many times we have to work with complex text and we would like these texts were much more understandable. Maybe just to get some keywords we serve to greatly shorten work. In the end, we find that spend too much time to understand the text. So it would be interesting to build an application that could easily extract certain parts of the text, analyze it to our liking and add that information to the text to understand it more easily.

For example, if we are studying some notes of History, it would be interesting to get the names of the text or even dates. This would have an overview of the text and could make a summary by extraction. For example:

“Cristobal Colón es famoso por haber descubierto América, el 12 de octubre de 1492, al llegar a la isla de Guanahani, actualmente en Las Bahamas..”

Table 2.1 shows the names and dates of this text we would get:

Cristobal Colón	América	12 de octubre de 1492
isla	Guanahani	Las Bahamas

Tabla 2.1: Result

Seeing these results, we can get a little idea of what the text is about and it could greatly helps the simplification process.

Another interesting example of use would be someone who wants to learn English vocabulary. It would be very useful, for example, to extract the text all adjectives or all names from the text, and quickly find all translations. Also, the person who is learning English would have all that enrichment available in their text, and could study it more easily.

Similarly, would also help learn Spanish vocabulary, as they may be synonyms or antonyms of words, definitions, or even a morphological analysis

of the words of the text.

Our goal in this project will be to do an application that can be used on any device, by anyone, that allows to do customized transformations in texts that help their simplification and understanding. Additionally, we add additional options such as morphosyntactic analysis, English translation, synonyms, antonyms and even definitions, which will offer many possibilities.

Already there are applications that do similar work, but our work will unite them all in one application that is easy to use, which has a user friendly appearance and can be run on different devices.

2.2. Objectives

In view of the needs that people have to understand written text, we have to meet the following objectives:

- Research other systems with the same or similar to the motivation of this project.
- Research available technologies and choosing the most suitable and research of useful web services to enrich the text.
- Design an efficient system architecture that combines the technologies chosen.
- Design a user-friendly interface validated by users and experts in the field of usability.
- Creating a web service to communicate with other web services that help to enrich the text. Construction of a client to communicate with the web service.
- System publication in a repository or web.

2.3. Document structure

Chapter 3 gives an overview of the technologies used and other systems in the field of text adaptation to facilitate their comprehension. We will discuss natural language processing, web applications and web services technologies, and systems related to the work presented.

In chapter 4 we discuss the process undertaken to define the system interface and the application behavior and how to interact with it. It also discusses what is the Goal-Centered Design, how it was adapted to this project and the development of a first prototype on paper and its validation with users.

Chapter 5 explains the system architecture, first with an overview of architecture and then introducing each of the components to be developed in later chapters in more detail.

Chapter 6 discuss the use of the application and its components along with its functionalities.

In chapter 7 we discuss how the server side is implemented. We explain the main service that communicates with the application, its operation, the call to this service and the steps carried out for the modification and enrichment of the text.

Chapter 8 presents the heuristic evaluations that we did with usability experts.

In chapters 9 and 10 we explain the conclusions that have been reached to develop the project and possible improvements or modifications that could be added in future iterations.

Finally in chapter 11 we explain the contributions from both members of the group in the course of the project.

Capítulo 3

Estado del arte

3.1. Procesamiento del Lenguaje Natural (PLN)

El procesamiento del lenguaje natural, también conocido como PLN o NLP (*Natural Language Processing*) en inglés, es un campo de la informática y la lingüística, que está relacionado con otros subcampos de la informática y la ciencia computacional, como la inteligencia artificial, el aprendizaje automático o la estadística. Su principal objetivo es hacer que una máquina pueda comprender el lenguaje de los humanos, y de este modo poder hacer interfaces de comunicación más naturales.

3.1.1. Análisis del lenguaje

El análisis del lenguaje consiste en determinar qué funciones tienen las palabras dentro del lenguaje, así como determinar cuál es la forma, clase o categoría gramatical de la misma. Según si estamos analizando la función o la forma de las palabras podemos distinguir dos tipos de análisis del lenguaje: análisis sintáctico y análisis morfológico.

En análisis sintáctico se encarga de ver qué función tienen las palabras dentro del lenguaje. Las palabras se agrupan entre sí para formar otras unidades más grandes como las oraciones, las frases o los sintagmas. Así, dependiendo de las relaciones de concordancia que tengan entre sí y la jerarquía que guarden entre ellas, las palabras adoptarán una función u otra.

Ver las funciones que tienen las palabras no siempre es un trabajo fácil ya que muchas oraciones son ambiguas, y su desambiguación escapa del alcance del análisis sintáctico. De este modo, a un nivel básico, podemos realizar el análisis sintáctico aplicando una serie de reglas fijas que permitirá asignar una estructura jerárquica a una oración. Esta estructura podrá ser diferente según qué reglas apliquemos.

Un ejemplo de análisis sintáctico podría ser el presentado en la Figura 3.1

para la frase “Juan lee el periódico en el autobús”.



Figura 3.1: Ejemplo de análisis sintáctico.

Por otro lado, el análisis morfológico se encarga de explicar la estructura interna de las palabras, es decir, la categoría gramatical de cada palabra. Dentro de estas categorías existen diversas subcategorías. Para recoger todas estas categorías, el grupo EAGLES ha propuesto un conjunto de etiquetas que llevan su nombre, para la anotación morfosintáctica de lexicones y corpus para todas las lenguas europeas. Estas etiquetas se pueden ver en la siguiente url: "<http://www.cs.upc.edu/~nlp/tools/parole-sp.html>".

La Tabla 3.1 recoge todas las categorías que puede tener una palabra:

Adjetivo	Adverbio	Artículo	Determinante	Nombre
Verbo	Pronombre	Conjunción	Numeral	Interjección
Abreviatura	Preposición	Signo de puntuación		

Tabla 3.1: Categorías gramaticales de las palabras

Un ejemplo de análisis morfológico podría ser el siguiente:

- **irá:** Verbo principal, modo indicativo, tiempo futuro, tercera persona del singular.

3.1.2. Analizadores lingüísticos

Maltparser (Hall, Nilsson y Nivre, 2007) es un generador de analizadores sintácticos de diversos idiomas. Esto es una tarea complicada debido a que cada lengua tiene sus propias reglas lingüísticas, y pueden diferir mucho de unas lenguas a otras. Además a este problema se le añade el problema de ambigüedad lingüística que tienen muchos lenguajes: Por ejemplo en el castellano, la palabra “ido” puede referirse al participio del verbo ir, o puede ser un adjetivo que describe a una persona distraída.

Lo que hace Maltparser ante estos problemas es usar técnicas de aprendizaje automático que generan, a partir de una gran cantidad de datos, un

modelo de lenguaje que va distribuyendo a cada palabra de una secuencia de palabras, una probabilidad. De este modo, cuando nos encontremos la palabra “ido” sabremos la probabilidad que tiene de que sea un participio o un adjetivo, y entonces el sistema podrá decidir qué tipo de palabra es más probable que sea. Esta probabilidad dependerá del resto del texto en que se encuentre la palabra.

Maltparser consigue este trabajo de manera bastante eficaz y eficiente, y suele ajustar muy bien las probabilidades de las palabras, debido a que ha sido entrenado rigurosamente. En concreto, para el castellano consigue un porcentaje de precisión que ronda el 80 %, un límite bastante aceptable para un lenguaje tan rico como el castellano.

Freeling (Chao, Carreras, Padró y Padró, 2004) es otro generador de analizadores lingüísticos multi-idioma que agrupa diversas funciones relacionadas con el análisis morfológico, sintáctico o léxico. Es una herramienta muy utilizada en aplicaciones de lenguaje natural como pueden ser la traducción automática o resumir texto, ya que guarda información sobre los lexemas, sus etiquetas, su probabilidad, información acerca de la forma de la palabra, información acerca de las frases, de los párrafos y del documento completo. Para procesar el lenguaje usa herramientas para identificar el idioma, devolver una lista con los tokens de los que se compone el texto, devolver las frases, analizar morfológicamente cada palabra, reconocer expresiones regulares, locuciones, números, expresiones temporales, signos de puntuación, expresiones de magnitudes físicas y monetarias, probabilidades léxicas, y nombres propios.

Veremos ejemplos de lo que hacen exactamente estas herramientas en la siguiente sección.

3.1.3. Herramientas de análisis a través de servicios web

En la sección anterior hemos visto varios analizadores lingüísticos muy usados a la hora de analizar texto. En esta sección entraremos en más detalle en las versiones web de estas herramientas y explicaremos un poco como se usan y lo que hacen.

Hasta ahora hemos tratado tanto el análisis sintáctico como el morfológico, pero también existen otras formas de analizar un texto como pueden ser la tokenización del texto, las concordancias y las dependencias entre palabras, estadísticas del texto, segmentación o el reconocimiento de entidades. Todas estas funciones las podemos tener fácilmente usando servicios web que ya tienen implementadas estas funcionalidades.

Cabe destacar el trabajo del Institut Universitari de Lingüística Aplicada (IULA), que ha recopilado muchos de estos servicios en su web, y que ofrece una versión web de estos servicios que es muy útil para analizar texto

rápidamente. Basta con introducir los datos necesarios y el servicio web se encarga de analizar todos los datos.

Como parte de nuestro proyecto, nos centraremos en hablar de los servicios web de IULA que más hemos utilizado y con los que más familiarizados estamos, y que tienen más que ver con el análisis sintáctico y morfológico, del que ya hemos hablado, y con los generadores de analizadores Freeling y Maltparser.

Para el análisis sintáctico de texto, IULA ofrece un servicio web de Maltparser con la funcionalidad de analizador de dependencias con nombre “malt_parser”. Este servicio crea un árbol que representa la relación de dependencia que tienen las palabras con el resto de palabras. De este modo podemos saber la categoría gramatical de la palabra viendo la relación que tiene con el resto de palabras.

Para usar la versión web de este servicio solo hay que introducir el texto que queremos analizar, o la URL donde está el texto que queremos analizar, introducir el idioma, y darle al botón “Run Service”. Esto generará una salida, un informe de cómo se ha ejecutado el servicio, y un estado que dice si se han encontrado fallos al ejecutar el servicio.

Un ejemplo de la salida de este servicio se puede ver en la Figura 3.2:

1	Esto	este	p	PD0NS000		2	SUBJ	-	-
2	es	ser	v	VSIP3S0	-	0	ROOT	-	-
3	un	uno	d	DI0MS0	-	4	SPEC	-	-
4	texto	texto	n	NCMS000	-	2	ATR	-	-
5	de	de	s	SPS00	-	4	MOD	-	-
6	prueba	prueba	n	NCF5000	-	5	COMP	-	-

Figura 3.2: Ejemplo de salida Maltparser.

En la salida podemos observar que la quinta fila corresponde al análisis morfológico de la palabra usando las etiquetas EAGLES, y en la columna siete vemos la palabra de la que depende esa palabra. Si ponemos estos datos en un árbol de dependencias quedaría como en la Figura 3.3:

Para el análisis sintáctico y morfosintáctico de texto, IULA ofrece varios servicios web basados en Freeling. Concretamente, nos centraremos en el servicio de etiquetado morfosintáctico de Freeling “freeling3_tagging”. Este servicio etiqueta las palabras usando las etiquetas EAGLES y añade la probabilidad de que el etiquetado sea correcto. Además también muestra información de donde empieza y donde acaba cada palabra del texto.

Al igual que para el servicio de Maltparser, para usar la versión web de este servicio solo hay que introducir el texto que queremos analizar, o la URL donde está el texto que queremos analizar, introducir el idioma, y darle al botón “Run Service”. Esto generará una salida, un informe de cómo

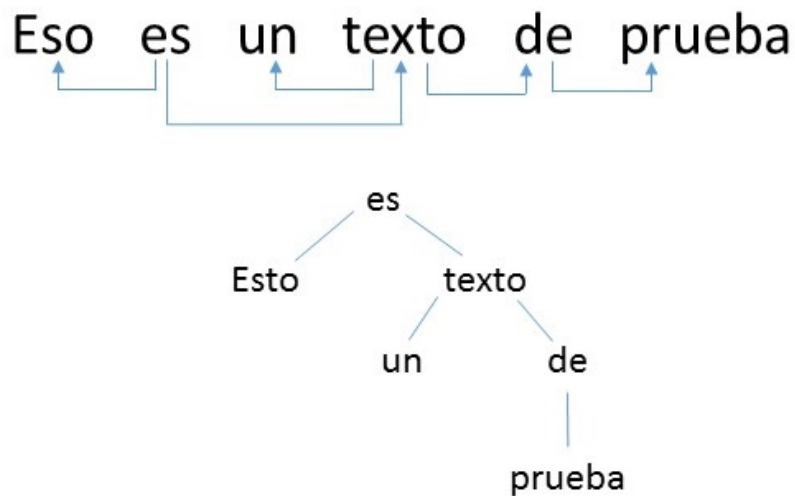


Figura 3.3: Árbol de dependencias de Maltparser.

se ha ejecutado el servicio, y un estado que dice si se han encontrado fallos al ejecutar el servicio.

Un ejemplo de la salida de este servicio sería el que muestra la Figura 3.4:

Esto	este	PDONS000	1	0	4
es	ser	VSIP3S0	1 5	7	
un	uno	DIOMS0	0.986987	8	10
texto	texto	NCMS000	1 11	16	
de	de	SPS00	0.999919	17	19
prueba	prueba	NCFS000	0.916667	20	26

Figura 3.4: Ejemplo de salida Freeling.

En la salida podemos observar que la tercera fila corresponde al análisis morfológico de la palabra usando las etiquetas EAGLES, en la cuarta la probabilidad de que sea acertada ese etiquetado, y en las columnas cinco y seis vemos respectivamente el inicio y fin de la palabra.

3.2. Tecnologías de servicios web

El término “servicio web” (Universidad de Vigo, 2008) (o *web service* en inglés) según el consorcio W3C (*World Wide Web Consortium*) (Berners-Lee y Jaffe, 1994) se define como “una aplicación software identificada por una URI (*Uniform Resource Identifier*), cuyas interfaces se pueden definir,

describir y descubrir mediante documentos XML. Los servicios web hacen posible la interacción entre aplicaciones software utilizando mensajes XML intercambiados mediante protocolos de Internet.” Además, distintas aplicaciones desarrolladas en diferentes lenguajes de programación y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos. Esto se consigue mediante el uso de mecanismos de comunicación estándar, que interactúan entre sí para presentar información de forma dinámica al usuario.

3.2.1. Protocolos de servicios web

Como se ha comentado anteriormente, los servicios web cuentan con un conjunto de servicios y protocolos denominado “pila de protocolos para servicios web” (o *web services protocol stack* en inglés). Esta pila de protocolos para servicios web está comprendida principalmente por cinco áreas (Alex Nghiem):

- Servicio de transporte: es el responsable de transportar mensajes entre las aplicaciones de red y los protocolos. El transporte de estos mensajes en la red se hace mediante los protocolos HTTP, SMTP, FTP y BEEP (*Blocks Extensible Exchange Protocol*), entre otros.
- Mensajería XML: se encarga de la codificación de mensajes en un formato común XML para que puedan entenderse extremos cualesquiera de una conexión de red. Este área incluye en la actualidad protocolos como XML-RPC (protocolo de llamada a procedimiento remoto), SOAP (*Simple Object Access Protocol*) y REST (*representational state transfer*) que se explicará en el siguiente apartado.
- Descripción del servicio: es el encargado de la descripción de la interfaz pública de los servicios web. Suele utilizarse WSDL (*Web Services Description Language*) para el formato de la interfaz. WSDL está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados. A menudo, se utiliza WSDL en combinación con SOAP ya que un programa cliente puede comunicarse con el servicio web y leer así el WSDL para determinar qué funciones están disponibles, y después utilizar SOAP para hacer las llamadas a las funciones que necesite.
- Descubrimiento de servicios: define cómo se anuncian y encuentran los servicios en la red. UDDI (*Universal Description and Integration*) es el protocolo que permite el descubrimiento de servicios web y lo hace centralizando los servicios en un registro común para ubicarlos y acceder a ellos.

- Seguridad para servicios web: define un conjunto de protocolos para aplicar seguridad en los servicios web. Comienzan por la especificación WS-Security que contiene especificaciones sobre cómo debe garantizarse la integridad y seguridad en mensajería de servicios web, incorporando características de seguridad en el encabezado de un mensaje SOAP. Existen seis especificaciones principales:
 - WS-Authorization: maneja el procesamiento de autorización para acceder a los datos e intercambiarlos.
 - WS-Secure Conversation: define cómo establecer una sesión protegida entre servicios para intercambiar datos usando las reglas definidas en WS-Policy, WS-Trust, y WS-Privacy.
 - WS-Policy: define las reglas de políticas sobre la interacción de servicios.
 - WS-Privacy: define cómo se mantiene la privacidad de las informaciones.
 - WS-Trust: define el modelo confiable para el intercambio seguro.
 - WS-Federation: define las reglas de identidad distribuida y de la gestión de esa identidad.

En la Figura 3.5 se muestra la estructuración de la pila de protocolos para servicios web.

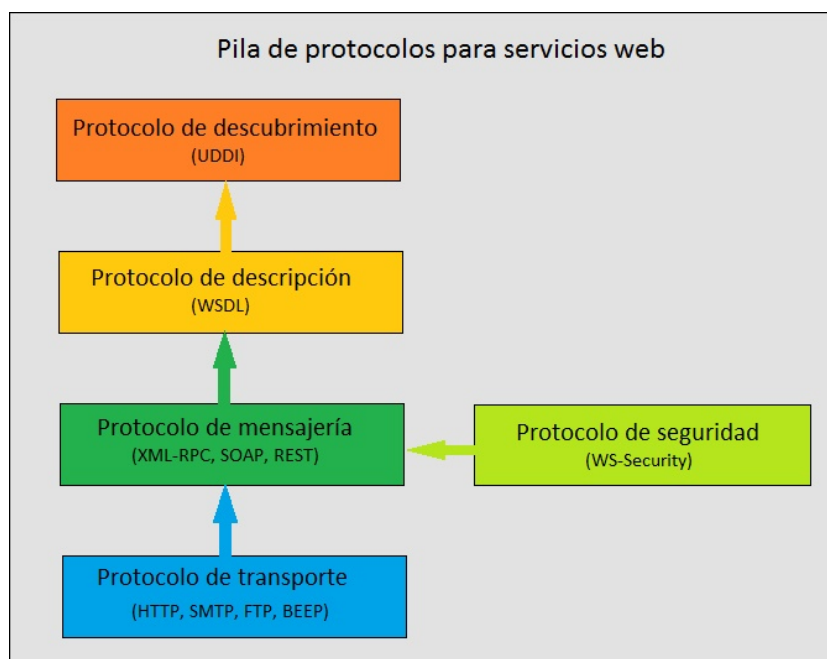


Figura 3.5: Estructura de la pila de protocolos para servicios web.

3.2.2. Transferencia de Estado Representacional (REST)

El término REST (*Representational State Transfer*) (Wikipedia, Representational State Transfer), es un conjunto de principios de arquitectura para sistemas hipermedia distribuidos como la World Wide Web. Es decir, un conjunto de reglas que define la interacción entre distintos componentes. Actualmente el término REST se utiliza en el sentido más amplio para describir cualquier interfaz web simple que utiliza XML y HTTP, sin abstracciones adicionales de protocolos de intercambio de mensajes como el protocolo SOAP mencionado en el apartado anterior.

Para que una aplicación sea REST debe cumplir una serie de reglas definidas (Alberto Fernández, 2013). Todas ellas vienen dadas en el protocolo HTTP:

- Arquitectura cliente-servidor: consiste en una separación clara entre dos agente independientes, un cliente y un servidor, en un intercambio de información.
- Stateless: el servidor no tiene por qué almacenar datos del cliente para mantener la comunicación.
- Cacheable: el servidor debe definir algún modo de cachear las peticiones que realiza el cliente.
- Sistema por capas: el cliente no tiene por qué saber por qué capas viaja la información, en otras palabras, el sistema no tiene que forzar al cliente a saber por qué capas tramita la información.
- Interfaz uniforme: la interfaz de comunicación debe ser común, es decir, no debe depender del servidor al que se hacen las peticiones y menos del cliente que es quién las realiza. Esta regla garantiza que da igual quien haga o reciba peticiones, siempre y cuando cumplan la interfaz definida previamente.

En la Figura 3.6 se muestra la arquitectura REST en base a estas reglas. Con todas estas reglas se podría implementar un protocolo para esta arquitectura pero, como se ha mencionado, el protocolo HTTP cumple todas ellas. Además, para que una aplicación sea completamente REST deberá implementar cuatro principios básicos. El protocolo HTTP implementa estos principios:

- Identificación de recursos: toda aplicación REST debe poder identificar sus recursos de manera uniforme. HTTP implementa esto con las URIs (*Uniform Resource Identifier*), es decir, la URL que usamos tradicionalmente aunque con una diferencia sutil ya que las URLs hacen referencia a recursos que, por lo general, pueden variar en el tiempo.

- Recursos y representación: como todo recurso debe tener una identificación (URI), REST define la manera en la que podemos interactuar con la representación del mismo directamente del servidor. HTTP define distintas cabeceras de tipos y un contenido en la respuesta, por lo que las aplicaciones pueden enviar el contenido en el formato que quieran, siempre y cuando este contenido contenga la información necesaria para poder operar con el objeto en el caso de que tengamos permiso para hacerlo.
- Mensajes autodescriptivos: el servidor debería devolver una respuesta que nos permita entender perfectamente cuál ha sido el resultado de la operación, así como si dicha operación es cacheable, si se ha producido algún error, etc. HTTP implementa esto a través de una serie de cabeceras y del estado.
- HATEOAS: incluir en las respuestas del servidor toda aquella información que necesita el cliente para seguir operando con el servicio web. Es decir, el cliente no tiene por qué saber que cuando obtenemos información tenemos además las opciones de modificarla o eliminarla, por lo que el servidor debe enlazar a estas operaciones en la respuesta a dicha petición. Así, un cliente sólo necesita saber el punto de entrada de la aplicación REST.

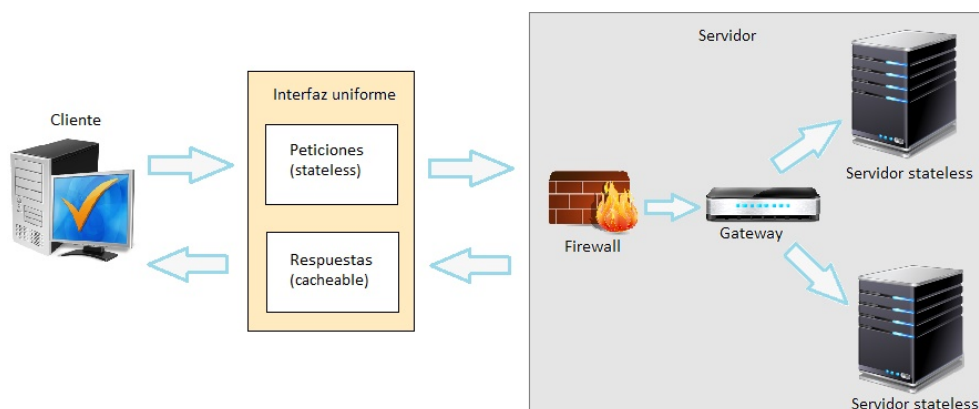


Figura 3.6: Arquitectura REST.

3.2.3. Servicios web RESTful

Los servicios web que siguen las reglas y principios propios de una arquitectura REST se denominan frecuentemente como RESTful. En otras palabras, un servicio web RESTful es un servicio web que implementa la arquitectura REST. Un servicio web RESTful contiene lo siguiente (Alberto Fernández, 2013):

- URI del recurso: por ejemplo, `http://servicio.es/recurso/coches/bmw`, daría acceso al recurso “coches” con marca “bmw”.
- El tipo de representación de dicho recurso: el servidor podrá devolver en la cabecera el campo “Content-type” seguido del formato en el que se presenta la información (HTML, XML, TXT, etc.) y así el cliente sabrá en qué formato devuelve el servidor la respuesta.
- Operaciones soportadas: HTTP define varios tipos de operaciones. Las siguientes operaciones son algunas de las más frecuentes:
 - GET: obtiene una representación de un recurso especificado.
 - POST: solicita que el servidor acepte la entidad adjunta en la solicitud como un nuevo subordinado del recurso web identificado por el URI.
 - PUT: crea el recurso identificado por el URI.
 - DELETE: elimina el recurso especificado.
 - CONNECT: convierte la conexión a una conexión segura mediante un túnel TCP/IP.
- Hipervínculos: la respuesta del servidor puede incluir hipervínculos hacia otras acciones que el cliente puede realizar sobre los recursos.

En la Figura 3.7 se muestra un ejemplo de una petición al servicio web de Youtube para que responda con la página web del vídeo solicitado.

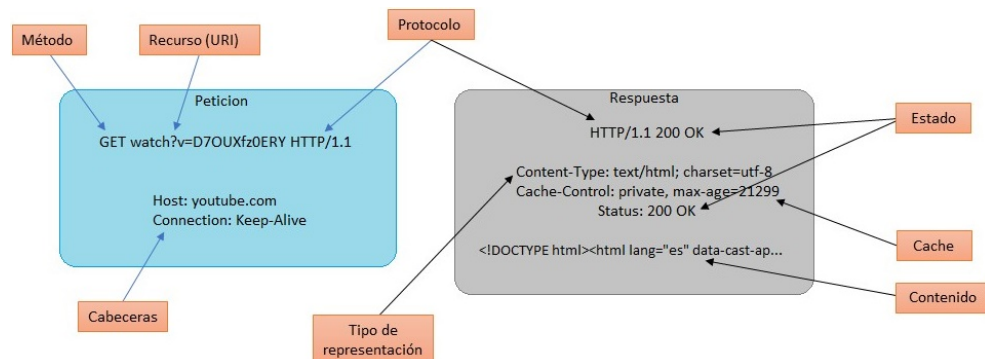


Figura 3.7: Ejemplo de petición GET al servicio web de Youtube.

En la actualidad, existen diversas APIs (*Application Programing Interface*) como JAX-RS que cuentan con la implementación de estos servicios web RESTful. También existen APIs para generar servicios web y clientes de servicios web con tecnologías SOAP y WSDL como JAX-WS. Ambas APIs están implementadas en Java y se explican a continuación en sendos apartados.

3.2.4. JAX-RS

Como se ha mencionado anteriormente, JAX-RS (Oracle, September, 2009) es una API implementada en el lenguaje de programación Java para la construcción de servicios web RESTful. JAX-RS proporciona una serie de anotaciones para cambiar una clase normal de Java a un recurso web. Las siguientes anotaciones son algunas de las más frecuentes:

- `@Path`: ruta relativa de acceso a una clase recurso o método.
- `@GET`, `@POST`, `@PUT`, `@DELETE`, `@HEAD`: tipo de operación HTTP para la petición de un recurso.
- `@Consumes`: tipo de representación de la petición.
- `@Produces`: tipo de representación de la respuesta.
- `@PathParam`: establece el parámetro como un segmento de ruta.
- `@QueryParam`: establece el parámetro como un valor de un parámetro de consulta.
- `@CookieParam`: establece el parámetro como un valor de cookie.

Existen varias implementaciones de JAX-RS como: Apache CXF, Apache Wink, JBoss, RESTeasy, Restlet y Jersey. Esta última es la implementación por referencia y es la que se utiliza en este proyecto.

3.2.5. JAX-WS

JAX-WS (GlassFish community) es una API de Java que simplifica mucho el trabajo de generar servicios web y clientes que se comunican con servicios web usando archivos XML. Estos archivos XML contienen información sobre la estructura del servicio, su codificación, información de cómo se invocan las funciones del servicio, y de cómo son las respuestas que devuelve el servicio. Para hacer servicios web con JAX-WS, el desarrollador definirá e implementará unos métodos en una interfaz usando el lenguaje de programación Java, que serán los que luego se podrán usar cuando se llame al servicio. Cuando un cliente quiera usar uno de estos métodos sólo tendrá que crear un objeto que represente al servicio, y luego sólo tendrá que invocar a los métodos que quiera, sin preocuparse por cómo están implementados por dentro. JAX-WS genera automáticamente los archivos XML necesarios para que se comuniquen el cliente y el servicio, ahorrándole así al desarrollador el tener que crearlos él mismo. Esto supone una gran ventaja ya que el cliente y el servicio podrían usar distintas plataformas Java y no pasaría nada, ya que JAX-WS usa tecnologías definidas en el W3C como HTTP, SOAP y WSDL.

3.3. Interfaces de usuario con HTML5

3.3.1. HTML5

HTML5 (W3Schools, c) es la quinta versión del lenguaje de desarrollo de páginas web HTML (*Hyper Text Markup Language*) que surgió con la World Wide Web y ha ido evolucionando hasta ahora. HTML5 está estandarizado y regulado por el consorcio W3C (World Wide Web Consortium).

HTML es un lenguaje de marcado, es decir, consta de texto (que será el contenido de la página web) y de marcas (conocidas como etiquetas o *tags* en inglés) que permiten dar formato al contenido de la página, estructurarla e indicar algún tratamiento especial para el texto. Además, permite incluir otros elementos como imágenes, vídeos, otros documentos, etc.

El objetivo de este lenguaje de marcado es estructurar el contenido de la página web con estas etiquetas. Además, se pueden asociar a cada etiqueta varios atributos para poder especificar características de formato, de tipo de información, etc. para que después sean procesadas por el navegador.

Las etiquetas de HTML se escriben con el siguiente formato:

`<marca>Contenido</marca>`

Es decir, el contenido está dentro del elemento “marca” el cual tiene una etiqueta de inicio y otra de cierre diferenciadas por “/”. El elemento “marca” se reemplaza por los elementos definidos en el lenguaje. En la Figura 3.8 se muestra un ejemplo.



Figura 3.8: Ejemplo de página web simple en HTML.

Por otro lado, todo documento HTML tiene que ir dentro del elemento “html”, es decir, entre las etiquetas “html” de inicio y cierre. Dentro de estas dos etiquetas se diferencian dos partes: cabecera (elemento “head”) y cuerpo (elemento “body”).

En la cabecera se especifican, entre otros, el título de la página web, metadatos como la codificación “utf-8” para poder visualizar texto con tildes o enlaces a otros archivos para que carguen junto con la página web como archivos CSS y Javascript que se explican más adelante.

En el cuerpo se encuentra la estructura y el contenido de la página web, definido con las diferentes etiquetas del lenguaje.

Por último, como se ha mencionado al principio de este apartado, HTML5 es la quinta versión de este lenguaje de marcado con lo que ofrece una serie de ventajas sobre las anteriores. Algunas de estas ventajas se enumeran a continuación:

1. Es nativo, es decir, no pertenece a nadie, es *open source*.
2. Simplicidad del código. Esto permite hacer páginas más ligeras que cargan con mayor rapidez.
3. Mayor compatibilidad con navegadores de dispositivos móviles.
4. Permite inserción de vídeos y audio de manera directa.
5. Posee la capacidad de ejecutar páginas sin estar conectado.
6. Nuevas capacidades JavaScript que aumentan la capacidad de almacenamiento.
7. Nuevas capacidades CSS3 como opacidad, transparencia, bordes redondeados, sombras, animaciones de transformación y transición, brillo, etc.
8. Permite realizar diseños adaptables a distintitos dispositivos (web, *tablets*, *smartphones*)

3.3.2. CSS3

CSS3 (W3Schools, b) es la tercera versión de las CSSs (*Coding Style Sheets*) u hojas de estilo en cascada. Al igual que HTML5, CSS3 está estandarizado y regulado por el consorcio W3C (World Wide Web Consortium).

Las hojas de estilo en cascada son archivos con extensión “.css” que definen la apariencia de una página web. Al estar centralizada en estos archivos, todas las páginas de un sitio web pueden compartir la misma hoja de estilos facilitando la gestión y apariencia del sitio web. Por otro lado, una página web puede incluir varias hojas de estilo, lo que da lugar a que algunos elementos tomen la apariencia de la combinación de dichas hojas de estilo.

El objetivo de estas hojas de estilo es separar la estructura del documento HTML de su presentación para no tener que modificar dicho documento al querer cambiar la apariencia. Además, permite aplicar incluso hojas de estilo diferentes dependiendo del dispositivo en el que se presenta la página web.

3.3.3. JavaScript

JavaScript (W3Schools, d) es un lenguaje de programación interpretado, es decir, es un lenguaje de programación en el que la mayoría de las implementaciones ejecuta las instrucciones directamente sin tener que hacer una compilación previa del programa a instrucciones en lenguaje máquina.

JavaScript fue desarrollado por Netscape Communications Corporation y más tarde estandarizado por ECMA Internacional (antigua European Computer Manufacturers Association) bajo el nombre de ECMAScript y se define como un lenguaje orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

El objetivo principal es utilizar este lenguaje en el lado del cliente, es decir, en la página web, para mejorar la interfaz de usuario, pudiendo hacer páginas web dinámicas que permitan ubicar elementos en un documento HTML y responder a eventos del usuario tales como el clic del ratón y navegación de páginas, entre otros. También existe una forma de JavaScript en el lado del servidor.

Todos los navegadores actuales interpretan este código integrado en las páginas web. JavaScript cuenta con una implementación del DOM (*Document Object Model*) para poder interactuar con una página web.

El DOM es la representación estructurada, en forma de árbol, de los elementos escritos en un documento HTML. En la Figura 3.9 se muestra un ejemplo sencillo de como sería el árbol DOM de una página simple.

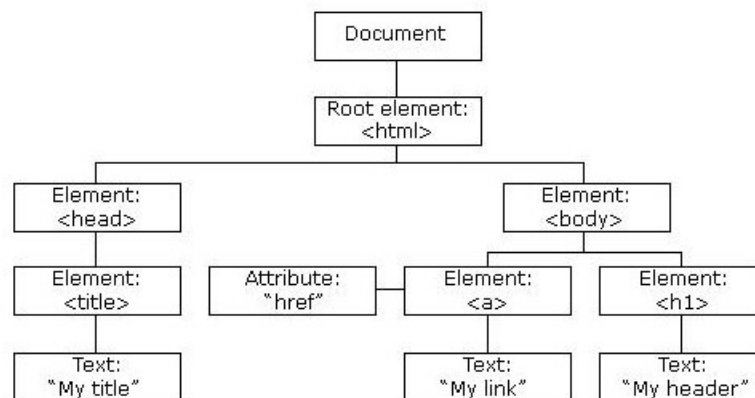


Figura 3.9: Ejemplo simple de árbol DOM.

Existen diversas bibliotecas (conjunto de implementaciones funcionales) de JavaScript, con licencia de software libre, como son jQuery y jQuery Mobile que se explican a continuación.

3.3.4. JQuery y JQuery Mobile

JQuery (W3Schools, e) es un conjunto de implementaciones funcionales creadas y lideradas por John Resig, programador y empresario de la Fundación Mozilla. La finalidad de esta biblioteca es la misma que cualquier otro documento JavaScript, es decir, permite interactuar con documentos HTML, manejar eventos, efectos, manipular el árbol DOM, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web, pero de forma mucho más simple ya que hay muchas funcionalidades desarrolladas que logran grandes resultados en menos tiempo y espacio.

JQuery permite cambiar el contenido de la página web sin tener que refrescarla mediante la manipulación del árbol DOM y peticiones AJAX que se explica a continuación.

AJAX (*Asynchronous JavaScript And XML*) (W3Schools, a) es una técnica de desarrollo web creada por Jesse James Garrett para crear aplicaciones interactivas que se ejecutan en el lado del cliente mientras se mantiene una comunicación asíncrona, en segundo plano, con el servidor.

JQuery Mobile (W3Schools, f) es un conjunto de herramientas, o framework, desarrollado y optimizado para dispositivos táctiles como *tablets* o *smartphones*. Actualmente sigue en desarrollo por el equipo de proyectos de jQuery.

3.3.5. Bootstrap

Bootstrap (Twitter) es un framework de software libre creado por Mark Otto y Jacob Thornton de Twitter para el desarrollo de diseño de sitios y aplicaciones web. Cuenta con diseño *responsive*, es decir, una aplicación o sitio web se adapta perfectamente a cualquier tipo de dispositivo independientemente de su tamaño. La versión actual se conoce como Bootstrap 3.

Está basado en los últimos estándares de desarrollo web (HTML5, CSS3 y JavaScript/jQuery) y cuenta con formularios, botones, plantillas de diseño con tipografías, menús de navegación, cuadros y más elementos de diseño.

3.3.6. Material design

Material design (Google) es un lenguaje visual desarrollado por Google que sintetiza los principios básicos de buen diseño. Material design tiene un diseño más limpio en el que predominan las animaciones y transiciones de respuesta, el relleno y los efectos de profundidad como la iluminación y sombras que proporcionan significado de lo que se puede tocar y cómo se va a mover. Cabe destacar que Google ha publicado una guía para la creación de interfaces de usuario con este diseño de materiales.

El sistema operativo Android Lollipop, la última versión de Android, cuenta con este diseño en su interfaz. Además el resto de aplicaciones que están en el Play Store de Google ya empiezan a utilizar esta guía para el diseño de sus interfaces. La Figura 3.10 muestra un ejemplo de la aplicación Whatsapp cuya interfaz en su última versión sigue este diseño de materiales.

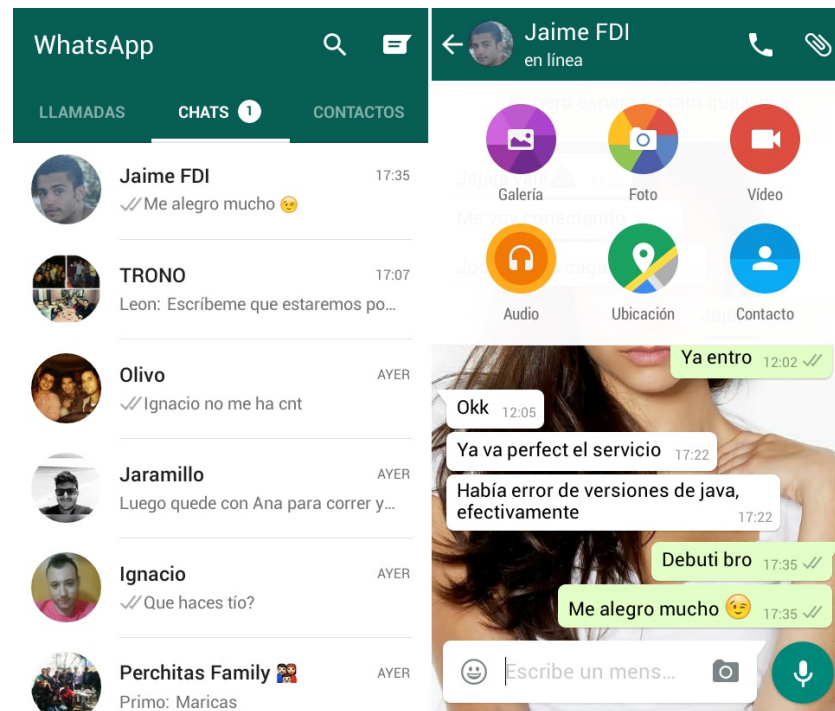


Figura 3.10: Interfaz de Whatsapp con Material design.

Actualmente existe una biblioteca de Material design para Bootstrap con licencia libre.

3.3.7. JSON

JSON (*JavaScript Object Notation*) (Ecma International) es un formato ligero para el intercambio de datos. Contiene un subconjunto de la notación que tiene JavaScript para los objetos. Actualmente se está utilizando este formato en lugar de XML en AJAX debido principalmente a que JSON tiene un formato mucho más sencillo.

JSON se basa en dos estructuras:

- Una colección de pares nombre/valor, es decir, una lista en la que cada campo contiene un par de elementos: uno el nombre del atributo y otro el valor asociado.

- Una lista ordenada de valores.

Estas dos estructuras de datos son universales en cualquier lenguaje de programación, lo cual quiere decir que es mucho más fácil implementar un analizador sintáctico (o *parser* en inglés) para poder obtener los datos necesarios.

Un ejemplo de este formato podría ser el de la Figura 3.11 en el que se tiene una persona con su nombre, apellidos, edad, números de teléfono y datos de su hijo.

```
var persona = {  
    nombre: "Paco",  
    apellidos: "Fernandez Garrido",  
    edad: 47,  
    telefonos: ["91456782", "678945812"],  
    hijo: {  
        nombre: "Antonio",  
        apellidos: "Fernandez García",  
        edad: 6  
    }  
}
```

Figura 3.11: Ejemplo de una persona en formato JSON.

3.4. Trabajos relacionados

Como se puede imaginar, todas estas tecnologías y herramientas pueden ser combinadas para la construcción de una página web cuyo objetivo sea la transformación (o enriquecimiento) de texto para facilitar al usuario la comprensión del mismo. A continuación veremos cuatro sistemas que tienen este objetivo y están implementados mediante la utilización de tecnologías web.

3.4.1. PorSimples

PorSimples (Aluísio, Specia, Pardo, Maziero y Fortes, 2008) es un proyecto que procesa los documentos en portugués disponibles en la web. Este proyecto persigue principalmente dos objetivos. El primero de ellos consiste en ayudar a niños que están aprendiendo a leer textos de diferentes géneros, a personas mayores con menos nivel académico, a personas con discapacidad

auditiva que quieren aprender portugués y a personas que estudian a distancia donde la comprensión del texto es de gran importancia. El segundo objetivo consiste en fomentar el área de investigación para el estudio de los problemas de comprensión de textos escritos.

PorSimples contiene diferentes formas de adaptación del texto como:

- Resúmenes automáticos: construcción de textos más cortos.
- Simplificación léxica: sustitución de palabras complejas por otras más simples.
- Simplificación sintáctica: degradación de oraciones complejas.
- Elaboración de texto: resaltado y adición de información de soporte.
- Un nuevo modelo de evaluación de legibilidad (o *readability assessment* en inglés).

Y cuenta con tres sistemas para el uso de estas herramientas:

- Un sistema llamado “FACILITA” que consiste en la adaptación de contenido web online. Este sistema es un *plug-in* que se añade al navegador web para resumir y simplificar contenido online. La Figura 3.12 muestra este sistema.
- Un sistema llamado “SIMPLIFICA” para la creación de texto simple. Este sistema consta de un editor donde se pueden crear textos simplificados. La Figura 3.13 muestra este sistema.
- Y un último sistema “FACILITA Educacional” para la elaboración léxica de textos web. La Figura 3.14 muestra este sistema.

3.4.2. NavegaFácil

NavegaFácil es un proyecto creado por dos estudiantes en la facultad de informática de la Universidad Complutense de Madrid como proyecto de fin de carrera (Bagó y García, 2013). El objetivo es muy similar al sistema FACILITA de PorSimples: facilitar la comprensión del texto contenido en páginas web pero en español. Las diferencias radican (aparte del idioma) en la implementación que tienen estos sistemas así como nuevas funcionalidades que incluye NavegaFácil.

NavegaFácil está implementado como una página web que actúa como si fuese un navegador web ya que en la página inicial consta de una cabecera en la que el usuario tiene que introducir la dirección de la página que desee.

Una vez abierta una página web, las funcionalidades que incluye NavegaFácil para facilitar la comprensión del contenido son las que se muestran en la Figura 3.15:



Figura 3.12: Sistema FACILITA de PorSimples.

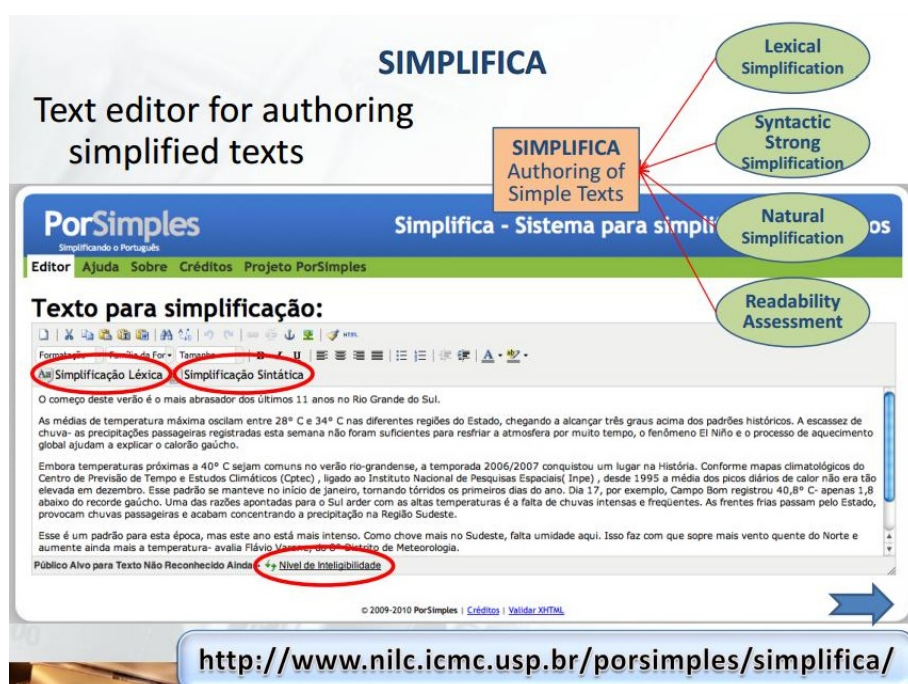


Figura 3.13: Sistema SIMPLIFICA de PorSimples.

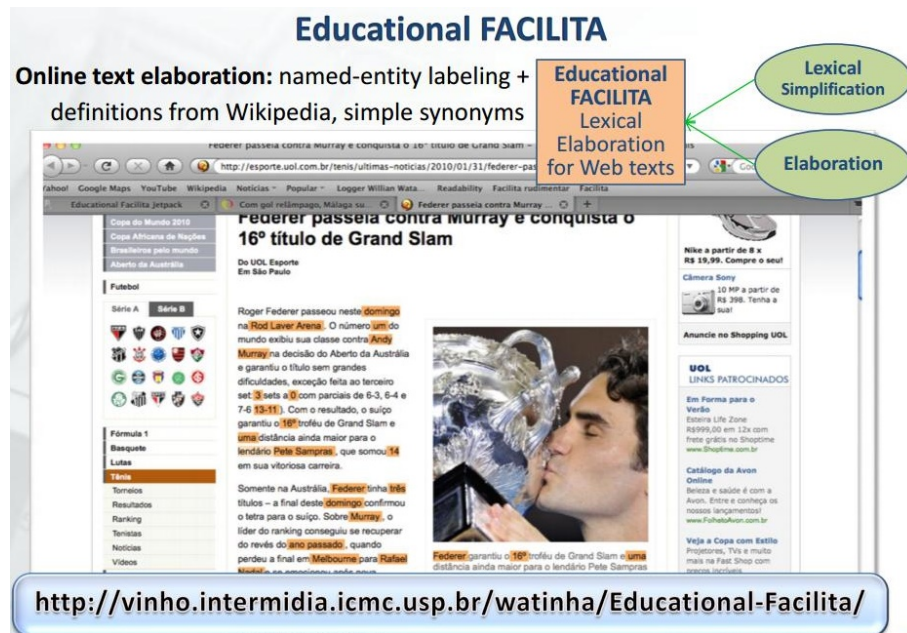


Figura 3.14: Sistema FACILITA Educacional de PorSimples.

- Buscar en Google.
- Mostrar definiciones.
- Reproducir en voz.
- Mostrar imágenes.
- Lematización.
- Sinónimos y antónimos.
- Traducir.
- Buscar en Wikipedia.

También incluye funciones para cambiar el estilo de la página así como las de guardar, cargar y otra de configuración donde se puede configurar el menú contextual, el idioma del traductor y otras opciones como refleja la Figura 3.16.

3.4.3. Open Book

Open Book es una herramienta desarrollada en el *FIRST project* (Barbu, Martín-Valdivia y Ureña López, 2013) con el objetivo de ayudar a personas con autismo que tienen niveles de coeficiente intelectual de 70 o más. *Open*



Figura 3.15: Funciones que ofrece NavegaFácil.

Book lleva a cabo este objetivo mediante la adaptación de documentos escritos en un formato que sea más legible y entendible para ellos. Este proceso de adaptación consiste en la reducción de la complejidad, la eliminación de



Figura 3.16: Configuración de NavegaFácil.

la ambigüedad y la mejora de la legibilidad e incluye las siguientes funciones:

- Sustitución de
 - Frases largas y complejas por varias frases cortas y sencillas.
 - Palabras largas o tecnicismos por palabras cortas y sencillas.
 - Lenguaje no literal por traducciones literales.
- Provisión de definiciones para palabras y frases ambiguas.
- Adición de
 - Imágenes.
 - Resúmenes concisos
 - Herramientas de navegación en documentos para documentos largos.

Esto permite que las personas con autismo puedan leer una amplia gama de documentos sin necesidad de ayuda alguna.

La herramienta Open book consiste en una página web donde podemos tener varios documentos asociados a una cuenta, y un editor que contiene una serie de funciones para adaptar texto como muestra la Figura 3.17.

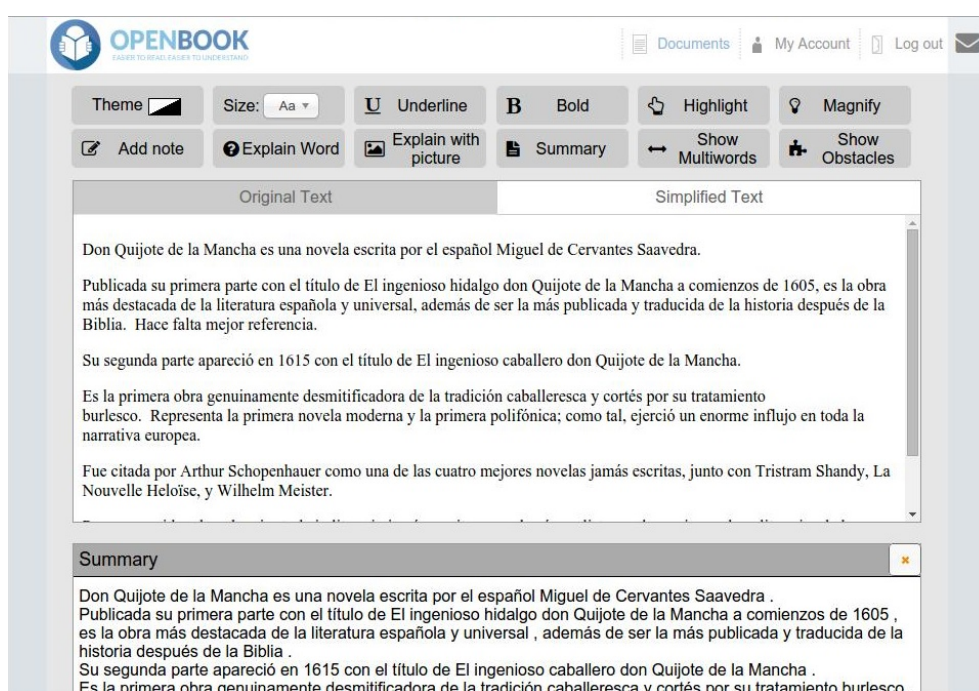


Figura 3.17: Editor Open book.

Capítulo 4

Diseño y requisitos

4.1. Diseño Guiado por Objetivos (DGO)

El Diseño Guiado por Objetivos, o DGO (Jiménez, Moreno y Sánchez, 2015), es una metodología de la ingeniería de la usabilidad propuesta por Alan Cooper que se basa en diseñar interacciones a partir de conocer muy bien al usuario y qué objetivos tiene a la hora de usar un sistema. Cuando se conocen los objetivos y el contexto en que va a ser usado el sistema, se elabora todo un proceso que nos conducirá al desarrollo final del sistema interactivo. Para ello se van a usar varios artefactos de los que hablaremos más adelante como pueden ser las personas, los escenarios o los *frameworks* de diseño.

Uno de los primeros conceptos que deben conocerse de esta metodología, y que tenemos que saber reconocer en los usuarios, son los objetivos. Un objetivo no es más que el propósito por el cual un usuario va a usar un sistema. No hay que confundir los objetivos con las tareas. Las tareas son el medio por el cual se va a alcanzar un objetivo. Por ejemplo, si una persona quiere ir de viaje a Nueva York, su objetivo es ir de viaje a Nueva York, pero el cómo ir, si ir en avión, en barco o como sea, que tenga que hacer antes la maleta, o que tenga que comprar el billete de avión, son tareas que tendrá que realizar para cumplir su objetivo. Los objetivos son los que hacen que las personas se comporten tal como lo hacen, y saber cuáles son los objetivos del usuario nos ayuda a entender por qué el usuario hace una tarea y cuáles son los pasos que realiza. De este modo podemos adaptar nuestro sistema a las necesidades del usuario y hacer que sea más adecuado para él. Podemos definir varios tipos de objetivos referidos al usuario del sistema:

- **Objetivos de experiencia:** Son los que dicen cómo se siente el usuario cuando interactúa con el sistema. Están muy relacionados con la estética del sistema, su diseño y las sensaciones que tiene el usuario cuando interactúa con él. Estos objetivos son muy importantes, ya que

si no los tenemos en cuenta, el usuario podría sentirse incómodo cuando usa nuestro sistema y podría no querer usarlo.

- **Objetivos finales:** Son los que dicen qué motiva al usuario a realizar una determinada tarea. Estos objetivos son los más determinantes en cuanto a la experiencia que tenga el usuario con el producto. Ejemplos: escuchar música, comprar, hablar con la familia y los amigos, etc.
- **Objetivos vitales:** Son las motivaciones más profundas que hacen que el usuario use nuestro producto. Ejemplos: tener éxito con algo, ser popular, ser un experto en algo, etc.

Hay otros objetivos que no tienen que ver con el usuario del sistema y que también tienen que tenerse en cuenta. Son los siguientes:


- **Objetivos del cliente:** Muchas veces el que compra el producto y el que lo va a usar no son la misma persona (un padre que le compra algo a su hijo, o una empresa que compra un producto para que lo use un empleado). Por esto, también tenemos que tener en cuenta al cliente, no sólo al usuario.
- **Objetivos de negocio y de organización:** Son los objetivos que tienen que ver con la productividad y la eficiencia de un negocio, o de mejorar una organización (por ejemplo, mejorar la educación).
- **Objetivos tecnológicos:** Son los objetivos que tienen que ver con el desarrollador del sistema. Por ejemplo, que el sistema sea multi-plataforma, que asegure la integridad de los datos, o que asegure la privacidad del usuario.

4.1.1. Fases del Diseño Guiado por Objetivos

El Diseño Guiado por Objetivos se puede dividir en las siguientes fases:

1. **Fase de investigación:** En esta fase se estudia al usuario potencial y/o real que va a usar el producto, y qué necesidades tiene. El diseñador es el que realiza esto con el fin de entender mejor al usuario, acercarse más a él y poder lograr empatía. En esta fase es donde se sacan los patrones de comportamiento que harán que podamos prever los objetivos y motivaciones del usuario. También se realizan otros estudios como los de mercado, auditorías y demás que ayudarán al diseñador a entender mejor el dominio, el modelo de negocio y las restricciones que tiene que cumplir el sistema. Algunas de las técnicas más comunes de esta fase son las entrevistas, cuestionarios, análisis de competencia, observación del usuario, grupos de discusión o análisis de datos de uso.

2. **Fase de modelado:** En esta fase se utilizan los datos de la fase anterior para crear modelos del dominio y de los usuarios. Concretamente, se crean las personas, que son modelos de usuarios que contienen información sobre los objetivos, motivaciones y los comportamientos de los usuarios a los que va dirigido el sistema (Figura 4.1). Al final de esta fase se tendrán los tipos de persona que representan a todos los usuarios del sistema, y que se usarán en la siguiente fase.



Ana Rodríguez:
 Persona no experta.

"Me encantaría poder programar la casa para que cocine o limpie mientras estoy fuera".

Demografía	Confort Tecnológico	Background personal
Grupo de edad: 35 – 55	Medio: Utiliza el móvil para llamar y chatear básicamente. Hace un uso normal de la tv, y usa pc para mails etc...	Ana Rodríguez, ama de casa de 42 años. Se dedica a sus labores, limpiar la casa, hacer la compra, hacer la comida, ocuparse de sus hijos, etc... En sus ratos libres da clases particulares de inglés a niños de colegio. Vive en las afueras, un piso medianamente grande, con su marido y dos hijos. Su marido se pasa casi todo el día fuera de casa trabajando, mientras que tiene que llevar y recoger a los niños al colegio.
Necesidades		
<ul style="list-style-type: none"> Ver de forma rápida el estado de los aparatos (encendido/apagado). Limitaciones que hagan segura la aplicación cuando él la use. 		
Motivaciones	Escenarios	Comportamientos
Apagar o encender las luces de la casa	<ul style="list-style-type: none"> En casa 	<ul style="list-style-type: none"> Interruptores.
Poner o quitar la alarma	<ul style="list-style-type: none"> En casa 	<ul style="list-style-type: none"> Dispositivo alarma
Encender el fuego	<ul style="list-style-type: none"> En casa 	<ul style="list-style-type: none"> Vitrocerámica
		<ul style="list-style-type: none"> Se levanta hasta el interruptor correspondiente y lo apaga. Se levanta hasta el dispositivo, y teclea la clave Enciende la vitroceramica y pone la temperatura correspondiente.

Figura 4.1: Ejemplo de persona.

3. **Fase de definición de requisitos:** En esta fase se utilizan las personas de la fase anterior, y todos sus datos, para crear los escenarios de contexto. Estos escenarios se usan para identificar los requisitos y necesidades del usuario. También se definen el resto de objetivos vistos anteriormente: requisitos de negocio, de clientes, requisitos técnicos, etc. A continuación, veremos un ejemplo de un escenario de contexto para una aplicación móvil que controla los diferentes dispositivos de una casa inteligente o domótica:

JUAN ÁLVAREZ SÁNCHEZ: Juan se levanta temprano para ir a trabajar y se prepara su café y la tostada como todos los días.

Cuando ha terminado de desayunar mete los platos en el lavavajillas y lo pone a funcionar, odia tener que hacerlo a la vuelta del trabajo. Al ir a trabajar ve unos cuantos mensajes graciosos de sus compañeros, parece ser que de la noche anterior. Al mirar el móvil ve que la temperatura exterior es bastante baja por lo que decide ir a la aplicación de Igloo y encender la calefacción en las habitaciones que va a frecuentar a su regreso, ya que no necesita calentar toda la casa, y especialmente la de su cuarto, ya que odia pasar frío por las noches. A la vuelta del trabajo enciende el horno para que al llegar ya esté caliente y pueda empezar a hornear esa cena que tanto le gusta y que tanto tiempo lleva de preparación en el horno. De camino a casa pasa por el supermercado y mira en Igloo la lista de alimentos que tiene en casa y se da cuenta de que le faltan unos cuantos productos que había pensado en comprar a la salida de casa, y el pan de molde para las tostadas de por la mañana.

4. **Fase de definición del *framework* de diseño:** En esta fase se crea el concepto general del sistema, definiendo como se va a comportar, su diseño visual, y su diseño físico. El *framework* de diseño se divide en otros tres tipos de *framework* que están muy ligados entre sí:

- *Framework* de diseño visual: se usa para definir los colores, la tipografía, los logotipos y cualquier otro elemento visual usado en la interfaz.
- *Framework* de diseño industrial: se usa para definir el diseño de los componentes hardware que se necesiten para la interacción con el sistema, si fuese necesario.
- *Framework* de interacción: se usa para definir la estructura de la interfaz, la organización de las pantallas, el flujo, el comportamiento y la organización del sistema. Para definirlo se puede realizar un proceso iterativo de seis etapas que llamaremos de la “a” a la “f”.

Estas son las seis etapas del proceso de definición del *framework* de interacción:

- a) Durante la definición del *framework* de interacción se define el factor de forma, la postura, y los métodos de entrada del sistema. El **factor de forma** consiste en averiguar si se puede presentar la información y el contexto en el que se va a presentar. Por ejemplo, para la interfaz de nuestra aplicación, tenemos que adaptarnos a las resoluciones de cualquier tipo de dispositivo sin importar su

sistema operativo (Android, IOs, Windows Phone, etc), y a cualquier tamaño de pantalla de ordenador ya que se podrá usar la versión web. Además, puede ser utilizado en cualquier lugar, por lo que hay que tener en cuenta cierto tipo de situaciones adversas como la luz del sol o la incomodidad de estar levantado usando la aplicación con una sola mano.

La **postura** se puede definir como la cantidad de atención que presta el usuario cuando interactúa con el sistema, y la cantidad de atención que tiene que prestar para interactuar con el sistema. Dependiendo de cuanta atención tenga que prestar al sistema podemos distinguir tres posturas diferentes:

- Una postura **soberana** es aquella en la que un usuario interactúa mucho tiempo, por lo que requiere una concentración mayor. La interfaz en estos casos tiene que ocupar toda la pantalla.
- Una postura **temporal** es aquella en la que un usuario utiliza el sistema de manera casual durante breves espacios de tiempo. La interfaz en estos casos tiene que ser lo más obvia posible y con grandes botones que indiquen qué acciones podemos realizar.
- Una postura **demonio** se usa en aquellos sistemas que se usan en segundo plano por lo que el usuario no interactuará con ellas casi nada. La interfaz en estos casos tiene que cumplir las mismas características que una postura temporal.

En nuestro caso, tendremos que adoptar una postura soberana ya que nuestra interfaz será una hoja en blanco en la parte central de la pantalla ocupándola por completo, donde se situará todo el texto que queramos enriquecer. Además, para poder enriquecerlo, también tendremos que poner las opciones que ofrecerá el sistema. Estas opciones también tienen una postura soberana ya que representan un flujo de trabajo.

Los **métodos de entrada** pueden ser un ratón, un teclado, un *trackpad*, una pantalla táctil, etc. Para nuestra interfaz usaremos como método de entrada una pantalla táctil en el caso de dispositivos móviles y ordenadores que tengan pantalla táctil, y un ratón y/o un teclado para los ordenadores.

- b) Para el *framework* de interacción también tendremos que definir los elementos y funcionalidades de la interfaz, que representarán a los requisitos de la fase de requisitos.
- c) Después hay que organizar estos elementos agrupándolos en unidades funcionales que faciliten a la persona hacer tareas similares.

- d) Cuando ya estén definidos todos los elementos y funcionalidades haremos unos bocetos básicos de cómo quedará nuestra interfaz.
- e) Después crearemos unos escenarios *key path*, que describirán como interactuará la persona con la interfaz. Estos escenarios pueden hacerse de manera narrativa, pero se recomienda que se creen de forma gráfica, usando prototipos en papel o mockups interactivos.

Podemos distinguir tres tipos de prototipos en papel: los guiones gráficos, los bocetos y los prototipos en papel interactivos.

Los guiones gráficos son una secuencia de dibujos rápidos y sin mucho detalle que relatan una historia concreta en la que un usuario persigue un objetivo y describe las tareas que el usuario realizaría para conseguir dicho objetivo. Es una especie de tira de cómic en la que se cuenta un escenario, y su principal ventaja es que no tenemos que centrarnos en una interfaz concreta.

Los bocetos son dibujos a más alto nivel, hechos a mano, de como sería la interfaz. Se hacen sin mucho detalle y sin centrarse en imágenes ni texto, solo nos centraremos en las funcionalidades. Generalmente, lo mejor es hacer muchos bocetos y coger las mejores ideas de todos ellos para hacer bocetos mejores.

Los prototipos en papel interactivos son maquetas físicas de la interfaz hechas a papel. Suele ser un fondo fijo de la aplicación, en la que se le van añadiendo componentes, todos ellos hechos a mano. De este modo, si se van cambiando los componentes se puede simular que la interfaz va cambiando. Se puede ver un ejemplo de prototipo en papel interactivo en la Figura 4.2.

Los prototipos en papel no ayudan a visualizar el diseño gráfico del sistema ni a ver qué sensaciones produce el sistema. Hay que tener en cuenta que un usuario no se va a comportar igual en la interfaz real que en la de papel. Por ejemplo en una interfaz en papel, el usuario se va a pensar más las acciones, y el tiempo de respuesta del sistema será diferente al de la interfaz real.

Para solucionar estos problemas podemos usar *mockups* interactivos. Estos son bocetos hechos a ordenador que simulan el cambio de los componentes cambiando de un boceto a otro a través de enlaces. Para esto hay muchas herramientas, desde editores de imágenes como Photoshop o Illustrator, hasta herramientas más específicas para este propósito como pueden ser Balsamiq o JustInMind. También se pueden usar presentaciones en PowerPoint o desarrollar prototipos en HTML. Para los prototipos HTML existen plantillas que ayudan a hacerlo más rápido.

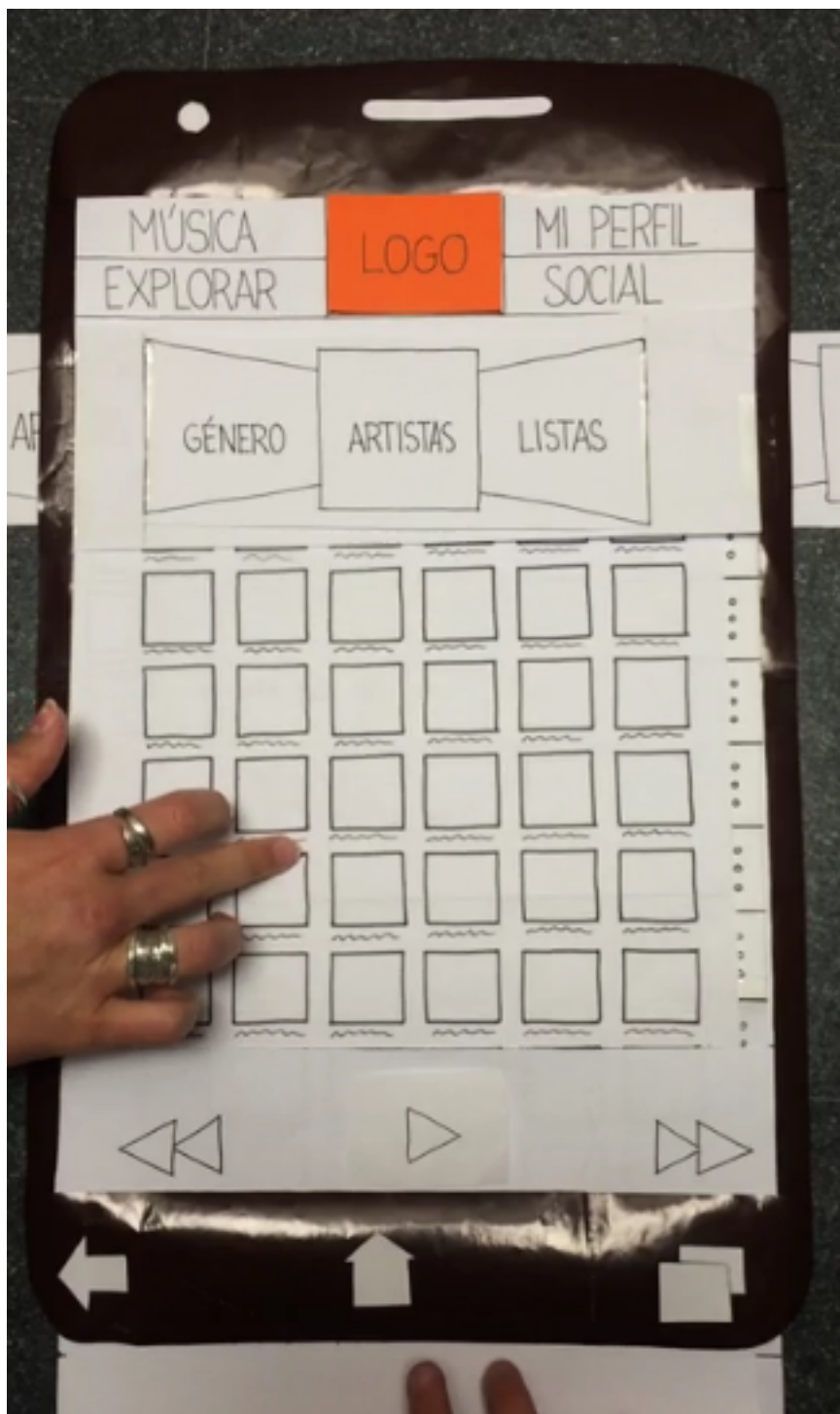


Figura 4.2: Ejemplo de prototipo en papel interactivo.

También podemos usar protipos Mago de Oz. Estos prototipos sirven para hacer pruebas en sistemas reales que todavía no están implementados. En este tipo de prototipo, el usuario interactúa con el sistema, y una persona “escondida” hace que el sistema se comporte como se espera. Para esto, es necesario tener la interfaz implementada tal como queremos que la vea el usuario, pero su implementación no puede ser tan costosa como crear el sistema real.

Otro método para hacer prototipos digitales, muy rápido de crear y barato, es hacer vídeos que describan como un usuario realizaría las tareas.

- f) Finalmente se valida el diseño creando escenarios que prueben el sistema y que ayuden a ajustarlo a las necesidades del usuario y a encontrar defectos.

Las etapas “c”, “d” y “e” se pueden intercambiar dependiendo del estilo del diseñador del *framework*: se puede partir de los escenarios para luego revisar las funcionalidades y realizar los bocetos, o hacer los bocetos para luego definir los escenarios y revisar las funcionalidades con los escenarios definidos.

5. **Fase de refinamiento:** En esta fase se cogen los *frameworks* de la fase anterior y de forma iterativa se evalúan y se validan hasta conseguir un diseño lo más detallado posible y con un grado alto de fidelidad. Además se realizarán estudios de usabilidad, y evaluaciones con usuarios para detectar problemas e ir refinando cada vez más el sistema.

Hemos hablado de evaluar los *framework*, pero hay que destacar que hay varios tipos de validación. De forma general, dependiendo del objetivo de la evaluación, distinguiremos dos tipos de evaluación:

- Si las evaluaciones se realizan para analizar un proceso, se realizan evaluaciones formativas o de diagnóstico. Estas evaluaciones se realizan en etapas intermedias del proyecto con el fin de guiar el diseño y encontrar problemas de usabilidad.
- Si se realizan para ver si un proceso está obteniendo los resultados esperados se harán evaluaciones sumativas o de verificación. Estas evaluaciones se realizan en las primeras etapas del proyecto o en las últimas para ver si el sistema es realmente usable.

Más concretamente, dependiendo de en qué momento se realice y dependiendo de los objetivos, podemos distinguir dos tipos de evaluaciones:

- Evaluaciones heurísticas: Son evaluaciones de bajo coste en tiempo y recursos, que se realizan con expertos en usabilidad, normalmente al principio del proyecto, que sirve para identificar rápidamente problemas de usabilidad sin tener que usar usuarios finales. También se suele usar al final del proyecto, en los casos en los que sea importante validar cuantitativamente la usabilidad de la interfaz.

Para realizar estas evaluaciones hay que informar al experto (o los expertos) de lo que hace el sistema. Después el experto tendrá que evaluar cada componente del sistema individualmente. Concretamente se fijarán en los 10 criterios de Nielsen o en las 8 reglas de oro de Shneiderman (Ver tabla 4.1 y tabla 4.2) y harán un informe en el que pondrán todos los elementos que no son usables, la heurística que violan y la severidad que creen que tiene ese problema. Además propondrán una forma de arreglar cada componente.

1	Visibilidad del estado del sistema
2	Relación entre el sistema y el mundo real
3	Control y libertad del usuario
4	Consistencia y estándares
5	Prevención de errores
6	Reconocimiento mejor que recuerdo
7	Flexibilidad y eficiencia
8	Estética y diseño minimalista
9	Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores
10	Ayuda y documentación

Tabla 4.1: 10 Criterios de Nielsen

1	Buscar la consistencia
2	Proveer accesibilidad para todos
3	Ofrecer feedback informativo
4	Diseñar el dialogo para aportar cierre
5	Ofrecer mecanismos simples de gestión de errores
6	Permitir deshacer acciones fácilmente
7	Permitir el Locus de Control
8	Reducir la carga de la memoria de trabajo

Tabla 4.2: 8 Reglas de oro de Shneiderman

Finalmente los desarrolladores usarán esta información para ver qué fallos tiene el sistema, les asignarán una prioridad usando las severidades que les han dado los expertos y su propio criterio, y

de este modo podrán arreglar los fallos de acuerdo a la prioridad resultante.

- **Evaluaciones con usuarios:** Son evaluaciones de alto coste en tiempo y recursos, que se realizan dejando interactuar a los usuarios con el sistema, para entenderles mejor. La información que se saca de estas evaluaciones es mucho mayor, pero es mucho más costoso.

Debido a este coste, lo mejor es preparar un buen plan de evaluación que sirva para guiar todo el proceso de evaluación. Esto suele conllevar hacer un formulario con preguntas específicas de investigación para los usuarios, las cuales tienen que ser lo más específicas posibles.

Después se especifica qué tipo de usuarios queremos que hagan la prueba. Deben reunir las mismas características que las personas del Diseño Guiado por Objetivos. Una vez que tengamos a los usuarios tendremos que preparar una lista de tareas que queremos que realicen, así como preparar el entorno donde se va a hacer la prueba y las herramientas que vamos a usar. Es preferible que el entorno nos deje ver las reacciones del usuario, pero que no se sienta intimidado por el proceso de evaluación.

La prueba se llevará a cabo con ayuda de un moderador que irá guiando al usuario para que realice todas las tareas y le animará a que exprese todos sus pensamientos cuando analice nuestro sistema. También tomará nota de todos los eventos relevantes que ocurran.

Usar un método u otro dependerá de la cantidad de recursos y tiempo del que dispongamos, así como de los objetivos que queramos cubrir.

6. **Fase de desarrollo:** En esta última fase, diseñadores de interacción y desarrolladores trabajarán conjuntamente para desarrollar el sistema y solucionar aquellas cuestiones que tengan que ver con la implementación de las interacciones.

4.2. Adaptación del Diseño Guiado por Objetivos para el proyecto

El diseño guiado por objetivos es una metodología bastante compleja, que conlleva muchas fases y cuya realización íntegra puede llegar a ser bastante costosa. Hay que tener en cuenta que conseguir usuarios para estudiar sus preferencias, y llegar a conseguir una empatía con ellos es una tarea bastante laboriosa y que requiere mucho tiempo, y en ocasiones también dinero. Además, para llegar a realizar bien esta metodología no basta con estudiar a

un par de usuarios, si no que se precisan un número aleatoriamente mayor y que variará con la dimensión del producto. También se necesitarán usuarios que prueben los prototipos, y teniendo en cuenta que estas pruebas se realizan iterativamente, y que es mejor que no se repitan los usuarios (ya que si conocen de antemano el sistema, los resultados no serán significativos), la cantidad de usuarios de prueba requeridos para realizar esta metodología puede llegar a ser muy alta. Sin embargo, es muy común no llevar a cabo exhaustivamente la metodología y adaptarla a las necesidades del proyecto, la cantidad de recursos y el tiempo disponible. En este apartado hablaremos sobre cómo hemos adaptado nosotros esta metodología a nuestro proyecto. Para ellos iremos recorriendo punto a punto todas las fases del desarrollo interactivo e iremos concretando cuál ha sido nuestra metodología:

1. **Fase de investigación.** En los últimos años ha habido varios proyectos finales de carrera de accesibilidad cuyo propósito era hacer más fácil la comprensión de texto de diferentes maneras. Por lo tanto la investigación en este dominio ha sido un trabajo que se ha ido realizando durante mucho tiempo. Nuestro trabajo en esta fase ha sido la de un análisis rápido de la competencia que nos permita ver qué nuevas funcionalidades podemos aportar a este campo. De este modo, hemos revisado los proyectos de nuestros compañeros de años anteriores, sobre todo Navega Fácil (Bagó y García, 2013), y además una gran variedad de sistemas con funciones similares a las que ofrece nuestro proyecto, como pueden ser *Open Book* (Barbu, Martín-Valdivia y Ureña López, 2013), *PorSimples* (Aluísio, Specia, Pardo, Maziero y Fortes, 2008). El análisis de estos proyectos se puede encontrar en la sección 3.4 del capítulo 3.
2. **Fase de modelado.** Esta fase es de gran importancia dentro del Diseño Guiado por Objetivos ya que nos permite comprender mejor al usuario y ver sus necesidades. Sin embargo en nuestro proyecto no hemos llevado a cabo una construcción exhaustiva de las personas, debido a que ya teníamos de antemano definido el tipo de personas a los que va dirigido nuestra aplicación, y aunque no los definimos formalmente, sí que las tuvimos en cuenta en todo momento a lo largo del proyecto.
3. **Fase de definición de requisitos.** En esta fase hay que utilizar las personas creadas en la fase anterior con el fin de identificar los objetivos y escenarios de contexto. En nuestro proyecto, al igual que en la fase anterior, no llevamos a cabo una definición exhaustiva de los requisitos ya que teníamos prefijados los objetivos principales que queríamos cubrir y sabíamos cuáles son los escenarios de contexto principales. Sin embargo, a lo largo del proyecto surgieron nuevos escenarios de contexto, que aunque no se reflejaron directamente en el proyecto, sí los tuvimos en cuenta indirectamente.

4. **Fase de definición del *framework* de diseño.** En esta fase se establecen los elementos funcionales de la interfaz, su comportamiento, el lenguaje y la forma en que se visualizan estos componentes. Además se usan los escenarios y requisitos de las fases anteriores para crear bocetos y prototipos de lo que será la interfaz del sistema. Nosotros creamos un prototipo en papel interactivo que fue validado una vez usando los escenarios que nos salieron de fases anteriores y, al final del proyecto, hicimos una evaluación con usuarios con el prototipo final.
5. **Fase de refinamiento.** En esta fase hay que refinar los *framework* de la fase anterior. Para ello lo mejor es ir haciendo pruebas con usuarios reales en las que se creen una serie de tareas que los usuarios tienen que hacer, y después hacer un *debriefing* con el participante para aclarar cualquier detalle del *framework*. Sin embargo, como esto es bastante costoso, nosotros lo que hicimos fue irlo refinando cada dos semanas según íbamos desarrollándolo, y al final, con el prototipo casi terminado hicimos una evaluación heurística con profesores de la asignatura “Desarrollo de sistemas interactivos” que se explicará más detalladamente en el capítulo 8.
6. **Fase de desarrollo.** Como ya hemos adelantado en la fase anterior, esta fase se hizo iterativamente junto con la fase anterior. Lo que hicimos fue desarrollar el prototipo cada dos semanas más o menos y refinar después de cada desarrollo. De este modo conseguimos adaptarnos a los cambios del sistema e ir refinando todos sus componentes.

4.3. Desarrollo del diseño en el proyecto

En las siguientes secciones describiremos como hemos desarrollado el proceso de Diseño Guiado por Objetivos en este proyecto. Comenzaremos con el diseño del *framework* y después se contarán las evaluaciones con usuarios que realizamos.

Ahora pasaremos a definir los elementos de datos y elementos funcionales que tiene nuestro sistema.

4.3.1. Elementos de datos y elementos funcionales

Los elementos de datos son los objetos y atributos de información que tienen que estar representados en el sistema para cumplir con las necesidades de información del usuario. Nuestro conjunto de elementos de datos, junto con sus atributos, puede verse de manera jerárquica como se muestra a continuación:

- Texto. Texto que el usuario dese enriquecer.

- Piezas. Elementos que identifican partes del texto.
- Acciones. Funciones que se aplican a las piezas para enriquecer el texto.
- Historial. Muestra todos los textos que el usuario enriqueció con anterioridad con la fecha y la hora. Consta de:
 - Eventos.
 - Texto enriquecido.
 - Fecha y hora de la última modificación.

Los elementos funcionales son las operaciones o acciones que pueden realizarse sobre los elementos de datos del sistema que pueden traducirse a acciones disponibles en la interfaz. Los elementos funcionales de este proyecto para los elementos de datos anteriores, son los siguientes:

- Texto
 - Copiar y pegar texto. Función para copiar y pegar texto.
 - Editar formato. Diversas funciones que permiten la edición del formato que tiene el texto.
 - Exportar e importar texto enriquecido. Funciones para guardar y cargar el texto que se ha enriquecido con las piezas y acciones del sistema así como dichas piezas y acciones aplicadas y el formato del mismo.
 - Aplicar y quitar piezas. Funciones para aplicar piezas al texto así como quitarlas.
 - Aplicar y quitar acciones. Funciones para aplicar acciones al texto así como quitarlas.
 - Guardar y Cargar textos con otra extensión (.txt, .pdf, etc). Funciones para guardar y cargar texto junto con su formato pero no las piezas y acciones que se aplicarán.
- Piezas
 - Crear piezas. Posibilidad para crear nuevas piezas.
 - Modificar piezas. Posibilidad para modificar piezas existentes.
 - Borrar piezas. Posibilidad para eliminar piezas.
- Acciones
 - Crear acciones. Posibilidad para crear nuevas acciones.
 - Modificar acciones. Posibilidad para modificar piezas existentes.
 - Borrar acciones. Posibilidad para eliminar acciones.

- Historial

- Ver eventos del historial. Visualizar todos los textos enriquecidos junto con la fecha y la hora de la última modificación.
- Mostrar un evento concreto.

4.3.2. Boceto del framework de interacción

Con el fin de saber cómo organizar todos estos elementos de datos y elementos funcionales, construimos un boteco en papel del *framework* de interacción.

Antes de construir este boceto se hicieron dos bocetos en sucio y de forma paralela (uno por cada integrante del proyecto) con el objetivo de poner en común las ideas y converger a un boceto bien definido ya que iba a ser evaluado por usuarios. La Figura 4.3 muestra la pantalla inicial de la posible interfaz del sistema en el boceto final.

Como este boceto iba a ser evaluado, hicimos todas las pantallas para las secuencias que iban a ser evaluadas con el objetivo de saber si la interfaz propuesta era usable. La Figura 4.4 y la Figura 4.5 muestran otras dos pantallas de este boceto.

La manera de interactuar con la interfaz propuesta en el boceto se cuenta en el siguiente apartado mediante la construcción de escenarios *key path*.

4.3.3. Construcción de los escenarios *key path*

Los escenarios *key path* son los que describen cómo interactúan los usuarios con la interfaz diseñada en el *framework* de interacción. Es decir, describe como interactúa una persona con los elementos de datos y funcionales de la interfaz. Los escenarios *key path* del proyecto son los siguientes.

La pantalla inicial se muestra como un simple editor de texto donde se puede copiar, pegar o escribir texto y cambiar su formato (Figura 4.3). En el momento que haya texto aparecerá, en la parte de abajo de la pantalla, una barra horizontal que indica las piezas y acciones que se pueden aplicar para enriquecer el texto. Al deslizar hacia arriba o pulsar el botón de la barra, aparecerá una pantalla donde se pueden aplicar o quitar piezas y acciones al texto.

Para cambiar entre piezas y acciones sólo hay que pulsar en las cajas de iconos o en las pestañas de abajo (Figura 4.4). Si se desea aplicar o quitar piezas o acciones solo hay que pulsar sobre ellas y se mostrarán en la caja de iconos correspondiente. Para eliminar una fila completa de piezas o acciones que está aplicada al texto, hay que desplazar dicha fila hacia la izquierda.



Figura 4.3: Pantalla inicial del sistema en el boceto.

Desde cualquier pantalla se puede acceder a los paneles laterales que están ocultos. El panel izquierdo muestra el historial de los textos que se han guardado con piezas y acciones aplicadas. El panel derecho muestra otras opciones:

- Abrir: para abrir un texto en formato .txt, .pdf, .ern, etc.
- Guardar: guardar el texto.
- Gestor de piezas y acciones: muestra en otra pantalla todas las piezas y acciones disponibles en la aplicación, pudiendo ver la información de

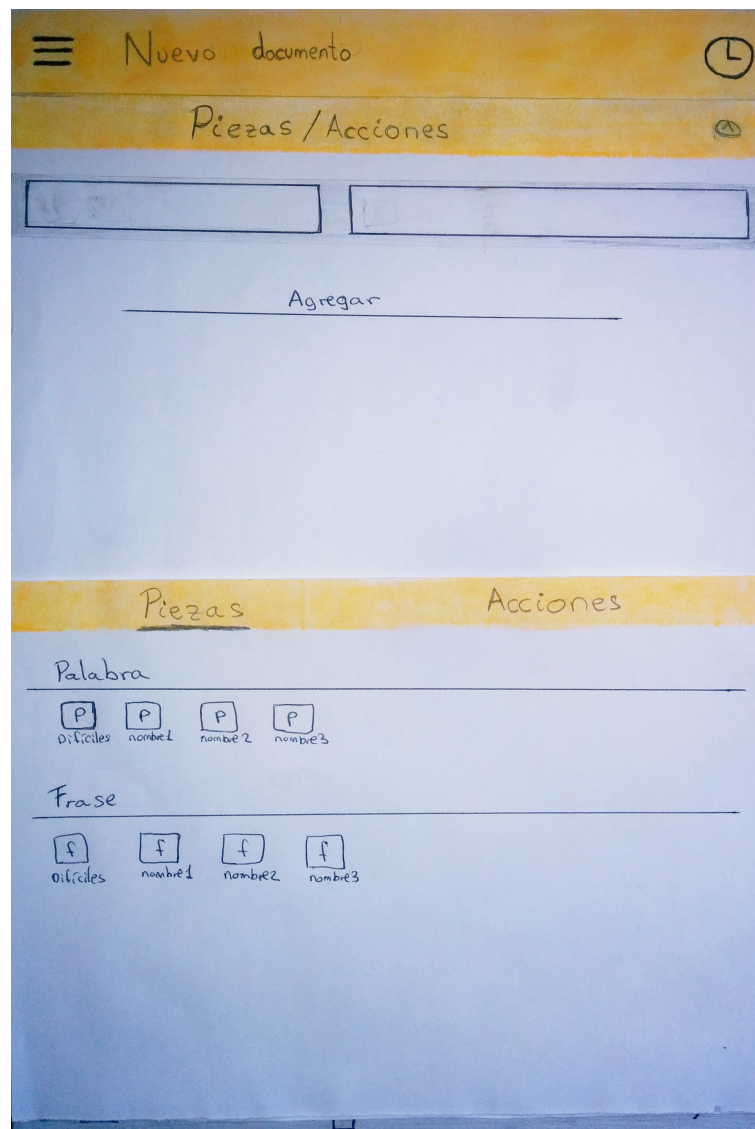


Figura 4.4: Pantalla para aplicar piezas y acciones

cada una al pulsar sobre ella, modificarlas, crear nuevas con el botón "+", seleccionarlas manteniendo pulsado en una de ellas y pulsando una sola vez en las demás y eliminarlas arrastrandolas al icono de la papelera o pulsando en dicho icono.

- Ayuda: muestra ayuda al usuario.
- Nuevo: nuevo documento en blanco.

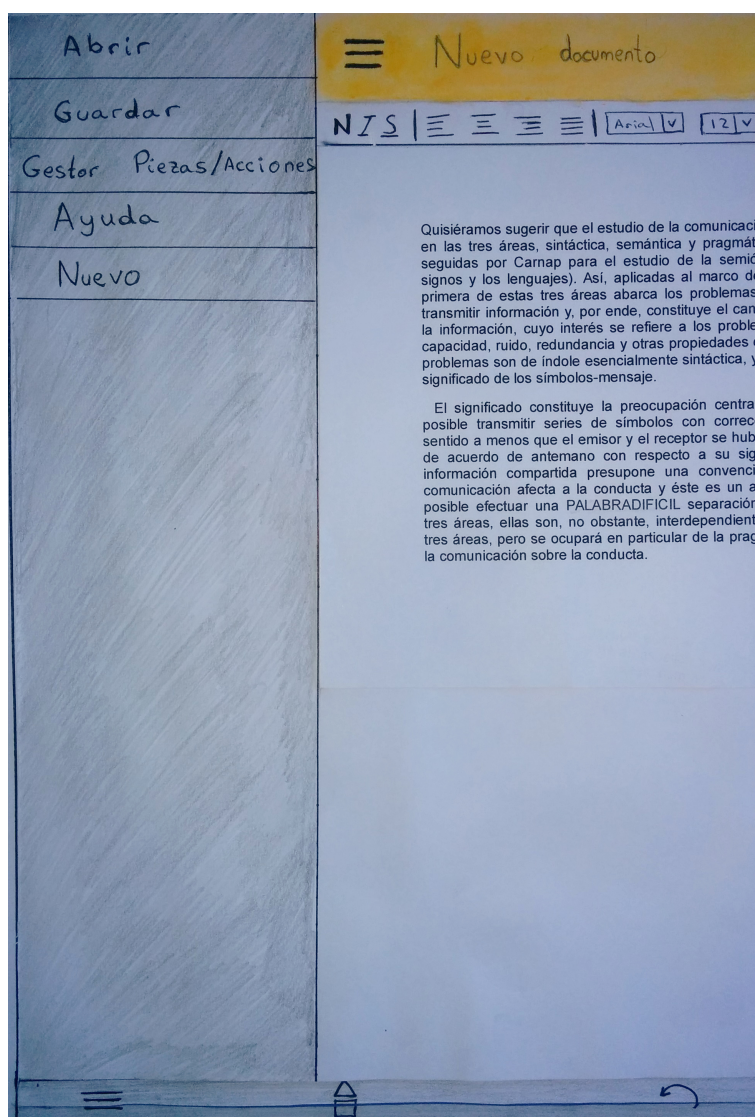


Figura 4.5: Panel lateral izquierdo con más opciones.

4.3.4. Validación del prototipo de baja fidelidad con usuarios

La sesión de evaluaciones se realizó el día 1 de diciembre en las instalaciones del usuario, es decir, en la Sala de Reuniones de la Facultad de Informática debido a que los usuarios eran nuestras tutoras. Se hicieron dos evaluaciones que no superaron los 30 minutos entre las dos. El perfil de usuarios que evaluamos fue el siguiente:

- Rango de edad: 25 - 35 años.
- Nivel académico: alto.

- Perfil tecnológico: experto.

Al comenzar la evaluación se dió una breve introducción a los participantes y una lista de tareas a realizar sin ayuda adicional (aunque el moderador pudo intervenir cuando el usuario se atascaba). Las tareas que tuvieron que realizar fueron las siguientes:

1. Escribir texto.
2. Aplicar las piezas “Palabras difíciles” y “Frases difíciles”.
3. Quitar la pieza “Frases difíciles”.
4. Aplicar la acción “Subrayar”.
5. Visualizar cómo ha cambiado el texto.
6. Guardar el texto.
7. Mirar el texto guardado en el historial de textos.
8. Eliminar las piezas y acciones aplicadas anteriormente.
9. Abrir el gestor de piezas y acciones.
10. Ver la información de la primera pieza.
11. Agregar dos piezas nuevas.
12. Eliminar esas dos piezas.

Tras realizar estas tareas se hizo una puesta en común (o *debriefing*) con cada uno de los usuarios por separado y después uno común para contestar a las preguntas que tuvieran, analizar los posibles fallos que tuviera la interfaz y proponer posibles soluciones. Además, toda la interacción con el prototipo fue grabada para registrar los momentos específicos en los que la interfaz resultó confusa y el moderador tomó notas. Estos vídeos de las evaluaciones pueden visualizarse en la dirección <http://bit.ly/1AUP1wt>.

Todo esto fue analizado de forma cualitativa para obtener el informe de hallazgos y recomendaciones que se detalla en el siguiente apartado.

4.3.5. Informe de hallazgos y recomendaciones

Como indica su nombre, un informe de hallazgos y recomendaciones consiste en un documento que resume los hallazgos encontrados (problemas encontrados) y propuestas sobre posibles soluciones. El informe de hallazgos y recomendaciones para este proyecto y después de haber realizado la sesión de evaluación es el siguiente:

- Las cajas de iconos relativas a las piezas y acciones no parecen interactivables, haciendo que el usuario no sepa que hacer cuando están vacías.
 - Recomendaciones: mostrar siempre las piezas y acciones para poder aplicarlas.
- El icono historial parece no ser muy intuitivo.
- La opción de eliminar todas las piezas y acciones que haya en una fila deslizando ésta hacia la izquierda no parece intuitiva.
 - Recomendaciones: hacer más visible la fila para que parezca que se puede interactuar con ella.
- Hay mucha confusión al abrir el “Gestor de Piezas/Acciones”. Se piensan que el gestor es donde se aplican las piezas y las acciones al texto.
 - Recomendaciones:
 1. Integrar el gestor (para agregar o eliminar) en el lugar donde se aplican las piezas y acciones. Esto puede ocasionar problemas si el usuario piensa que esas piezas solo se agregarán o eliminarán exclusivamente para ese texto y no en la aplicación.
 2. Cambiar el nombre de “Piezas/Acciones” por otro más significativo como por ejemplo “Enriquecer texto” y mantener el gestor aparte.
- No se puede volver al documento que está abierto desde el “Gestor de Piezas/Acciones”.
 - Recomendaciones:
 1. Añadir una nueva funcionalidad, “Borradores”, que esté en el menú opciones y se guarden ahí los documentos no guardados.
 2. Agregar un botón “Aceptar” en la esquina inferior derecha del gestor para regresar al documento.
- La opción de multiselección para las piezas y acciones del gestor, manteniendo pulsado en una de ellas y pulsando una vez en las demás, no parece intuitiva. Quizás porque es un prototipo en papel y no se muestra feedback para ello.
- Añadir una nueva funcionalidad “Guardar como...” en el menú opciones para poder guardar en el formato que desee el usuario.

Capítulo 5

Arquitectura

El esquema general de la arquitectura del sistema se presenta mediante una serie de figuras y una descripción para cada uno de los distintos componentes. Empezaremos primero con una visión general y muy básica de la arquitectura y después profundizaremos en cada una de las partes en los siguientes apartados.

La Figura 5.1 muestra esta arquitectura básica y general, así como la comunicación entre los distintos componentes.

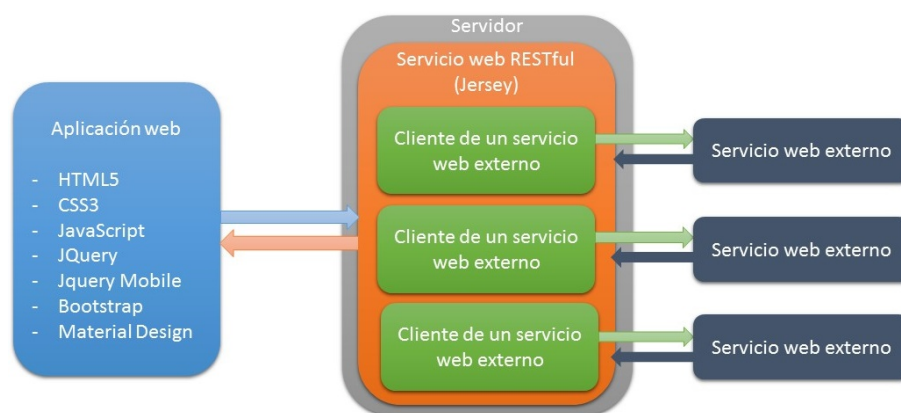


Figura 5.1: Arquitectura y comunicación del sistema

Como se aprecia en la figura, la arquitectura del sistema se compone de una aplicación web que se comunica con un servicio web RESTful mediante el envío de una serie de parámetros en peticiones HTTP. Este servicio web está alojado en un servidor y es el encargado de procesar las peticiones que llegan y de llamar a los clientes necesarios de otros servicios web externos dependiendo de los parámetros recibidos en la petición HTTP.

Los clientes se comunican con los servicios web externos según la implementación que estos tienen. Estos servicios web devuelven la respuesta

a nuestro servicio web RESTful, el cual procesa dicha respuesta y envía la respuesta definitiva a la aplicación web.

A continuación se detalla la arquitectura de cada una de las partes mediante diagramas UML y su respectiva explicación.

5.1. Aplicación web

La aplicación web está compuesta por el paquete UI. La Figura 5.2 muestra el diagrama UML de este paquete.

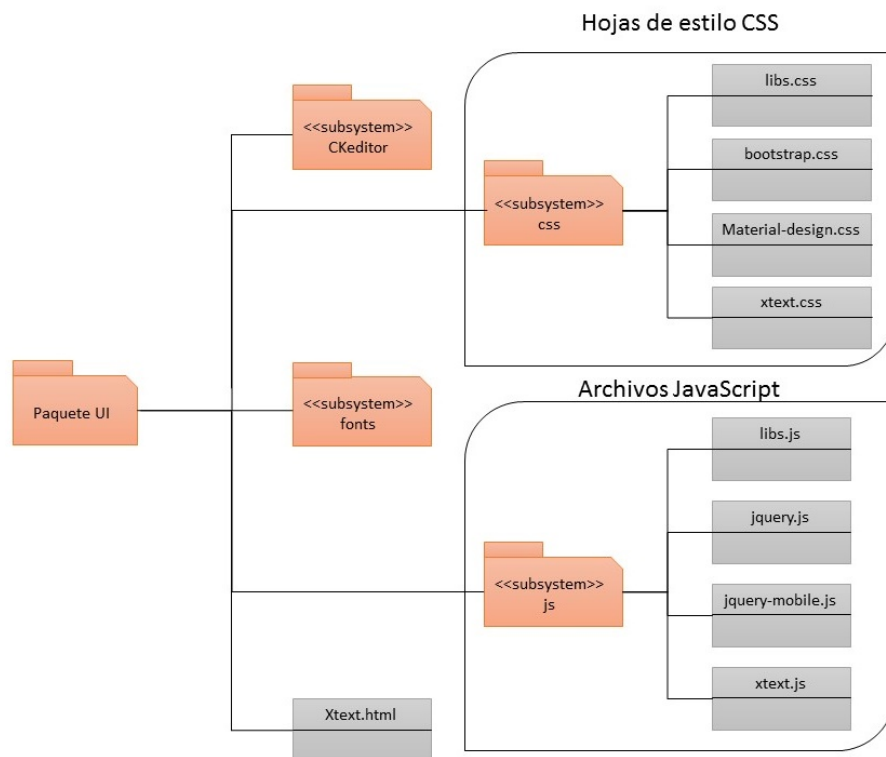


Figura 5.2: Arquitectura de la aplicación web

Este paquete contiene todos los archivos referentes a la aplicación web así como el cliente para la comunicación con el servicio web. Estos archivos son los siguientes:

- Paquete CKEditor: contiene el editor CKEditor de la aplicación en el que se muestra el texto así como su formato.
- Paquete css: contiene las diferentes hojas de estilo que se usan en el proyecto. Entre ellas se encuentran las css de Bootstrap, Bootstrap con

Material design y xtext.css que es la hoja de estilo que define el diseño de la interfaz de la aplicación web.

- Paquete fonts: contiene las fuentes e iconos que tienen Bootstrap y Bootstrap con Material design.
- Paquete js: contiene los diferentes archivos JavaScript que se usan en el proyecto para proporcionar dinamismo a la aplicación web. Entre ellos se encuentran los archivos jQuery y jQuery Mobile que contienen diferentes funciones para tal fin y para la comunicación con servicios web mediante la técnica AJAX. EL archivo xtext.js hace uso de estas funciones para que la aplicación web sea dinámica y contiene el cliente para la comunicación con el servicio web.
- Xtext.html: contiene la estructura y los diferentes elementos que aparecen en la aplicación web.

5.2. Servidor

Como se ha ido contando hasta ahora, el servidor es un servicio web RESTful que está implementado en Java con la API de Jersey. La arquitectura de este servicio web está compuesta por el paquete SRC. La Figura 5.3 muestra el diagrama UML de este paquete.

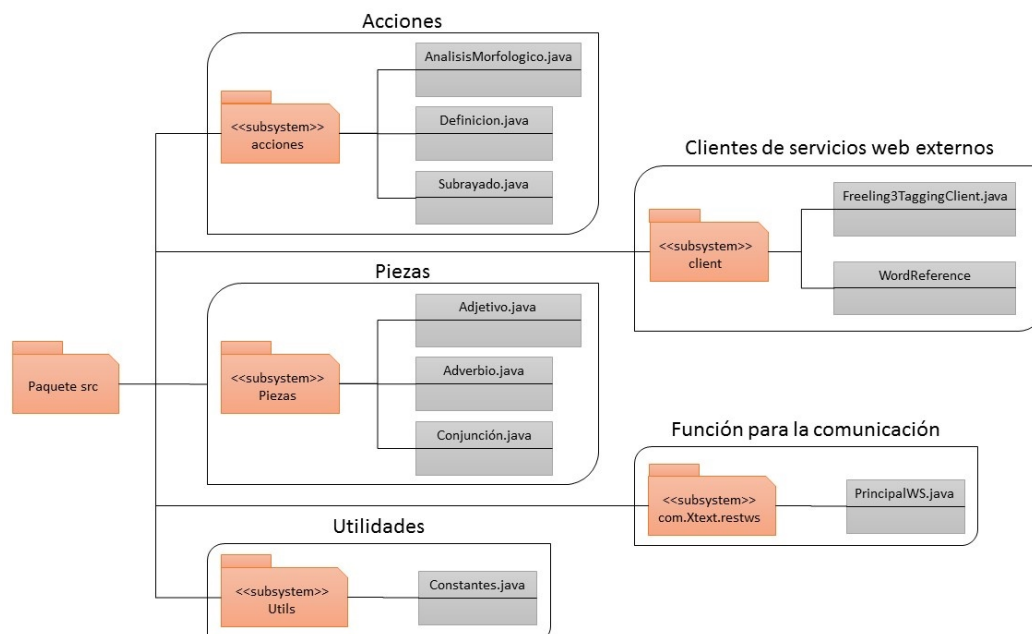


Figura 5.3: Arquitectura del servicio web

Este paquete contiene los archivos referentes al servicio web donde se incluyen los archivos para la comunicación, forma de uso y obtención de resultados de servicios web externos entre otros. Estos archivos son:

- Paquete de acciones: contiene las acciones que puede realizar el sistema. Tanto las que puede usar el usuario con la interfaz, como las que se usan para ayudar al procesamiento del texto.
- Paquete de piezas: contiene todas las piezas que puede tener un texto, es decir, todos los posibles tokens.
- Paquete de clientes: contiene todos los clientes que usan servicios web externos.
- Paquete `com.Xtext.restws`: contiene el servicio REST que es el que hace toda la funcionalidad del sistema. La comunicación con este servicio se realiza utilizando la API de Jersey.
- Paquete de utilidades: está compuesto por un archivo que contiene diferentes constantes de piezas para su identificación en las peticiones HTTP y etiquetas que se añaden al texto para que sean visualizadas en el editor.

Capítulo 6

Aplicación para el enriquecimiento personalizado de textos

La aplicación es una web desarrollada en HTML5, CSS3 y JavaScript. Se han utilizado las APIs de JQuery y JQuery Mobile para la implementación de las animaciones y el cliente para la comunicación con el servicio web implementado en el servidor. También se han utilizado las hojas de estilo de Material design de Bootstrap para implementar la interfaz de la aplicación.

Es una aplicación web multiplataforma, es decir, se puede utilizar en cualquier dispositivo ya sea PC, *tablet* o *smartphone*. Esto conlleva que el diseño de la interfaz debe adaptarse a diferentes tamaños de pantalla, es decir, debe tener un diseño adaptativo (o *responsive*). Este diseño adaptativo se ha conseguido con la utilización de los elementos *responsive* de HTML que ofrece Bootstrap.

El sistema consta de un editor de texto que contiene las funcionalidades usuales de cualquier editor, añadiéndole diferentes herramientas (o funcionalidades) con el objetivo de enriquecer el texto y facilitar la comprensión del texto introducido por el usuario. Las funcionalidades se basan en aspectos tales como la semántica del texto y el formato del mismo. Estas funciones generan acciones visuales que quedan reflejadas en el texto mediante la modificación del formato de palabras, o adición de nuevo contenido.

En el siguiente apartado se presenta el uso de la aplicación, los elementos que la componen, la forma de interactuar con dichos elementos y las herramientas que ofrece el sistema para enriquecer texto.

6.1. Uso de la aplicación y elementos que la componen

Esta aplicación permite a los usuarios enriquecer los textos que ellos deseen utilizando las diferentes herramientas que se ofrecen con el fin de facilitar la comprensión de dichos textos. El acceso al sistema se puede hacer de dos formas:

- Introduciendo en la barra del navegador de cualquier dispositivo la siguiente dirección:
<http://sesat.fdi.ucm.es:8080/xtextws/UI/Xtext.html>
- Instalando la aplicación en un dispositivo con sistema Android.

Una vez accedido al sistema se muestra la página principal. La Figura 6.1 muestra el aspecto visual de la aplicación que se cuenta a continuación.

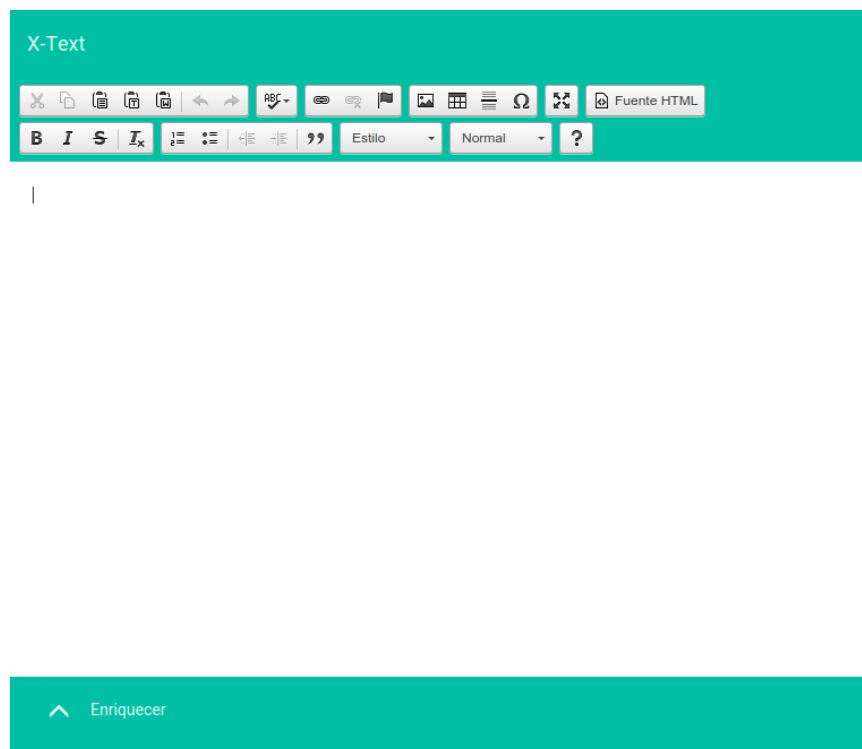


Figura 6.1: Página principal de la aplicación web.

Como se aprecia en la figura, la página principal consta de una barra superior fija donde se encuentra el nombre del sistema, un editor con licencia libre llamado CKEditor que ofrece una serie de herramientas para modificar el formato de textos, y una barra inferior que oculta la página de enriquecido

donde se encuentran las herramientas ofrecidas por el sistema para enriquecer textos.

6.1.1. CKEditor

Como se acaba de comentar, CKEditor (CKSource) es un editor de texto para aplicaciones web que cuenta con una licencia libre con lo que cualquier persona puede utilizarlo y modificarlo a su gusto. CKEditor está implementado en HTML y JavaScript y para poder incorporarlo en una aplicación web basta con descargar la implementación desde la página web www.ckeditor.com, elegir entre las tres versiones que ofrece (básica, estándar o completa), incluir los archivos en el paquete de la aplicación y llamar desde un archivo JavaScript a las funciones necesarias para incluir este elemento en el archivo HTML de la aplicación.

La diferencia que existe entre las tres versiones radica en el número de herramientas (o funciones) que ofrecen para dar formato a textos. La Figura 6.2 muestra la versión estándar que es la que se ha utilizado en el sistema.

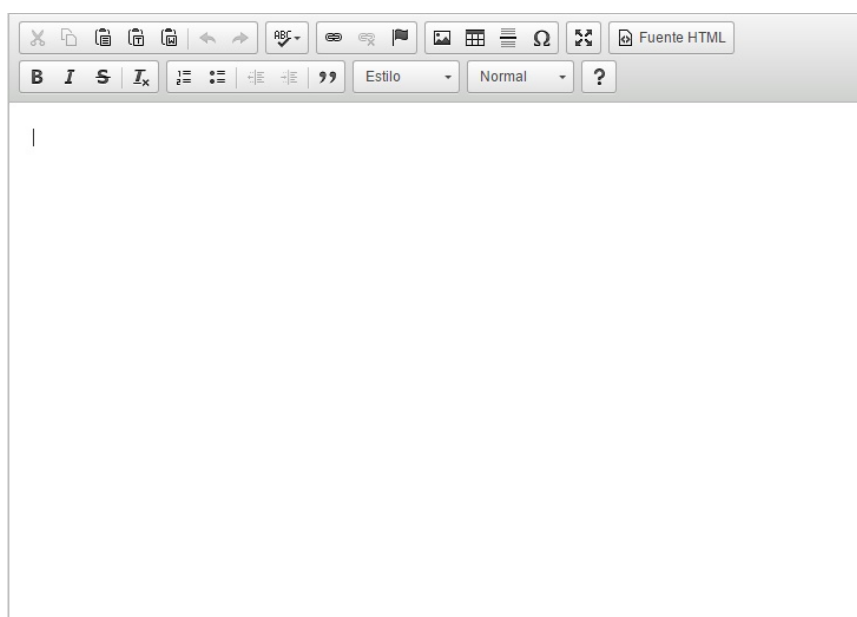


Figura 6.2: Versión estándar de CKEditor.

Esta versión incluye las herramientas usuales de cualquier editor tales como negrita, cursiva, tachado, copiar, pegar, cortar, deshacer, rehacer, corrector ortográfico, estilos para párrafos y caracteres, formatos para encabezados, numeración, inserción de imágenes, tablas y caracteres inusuales, entre otros.

Para explicar el funcionamiento de CKEditor primero empezaremos ex-

plicando el motivo que nos llevó a utilizar un editor ya implementado en lugar de implementar algo similar con HTML y JavaScript.

El motivo principal del uso de CKEditor en el sistema fue la inexistencia de elementos HTML que permitan introducir formato en campos o áreas de texto. Es decir, HTML tiene elementos para introducir texto, modificarlo y borrarlo pero estos elementos solo permiten texto sin formato. Esto originó un gran problema para el proyecto dejando como posibles soluciones la inserción de elementos HTML en la página web para dar formato al texto (inviable ya que el usuario no podría modificar el texto), implementar un editor mediante la modificación de un elemento HTML para tal fin o la posible incorporación de un editor implementado en HTML.

La implementación de un editor se basa en la modificación de un elemento HTML existente para que admita texto con formato mediante la inserción de elementos HTML en dicho elemento, es decir, insertar elementos HTML que dan formato a texto en páginas web dentro de otro elemento que permite la inserción de texto sin formato. Es una tarea compleja que requiere mucho tiempo y de haberlo implementado habríamos dejado de lado el objetivo principal de la aplicación.

CKEditor lleva a cabo esta implementación, modifica el elemento “textarea” existente en HTML que permite texto sin formato y lo convierte en una subpágina web donde se pueden introducir elementos HTML que dan formato a texto como negrita (“texto”), cursiva (“texto”) y demás elementos que se han mencionado antes como herramientas.

Esta idea hace posible que un usuario pueda dar formato a un texto mediante la utilización de elementos HTML el lugar de utilizar los botones que ofrece CKEditor, lo que facilita el aprendizaje de este lenguaje.

Para concluir, hay que mencionar que CKEditor no tiene diseño *responsive* y dificulta su visualización en dispositivos móviles aunque ya están trabajando en ello.

6.1.2. Enriquecer texto

En la interfaz hay una segunda página donde se encuentran disponibles todas las herramientas para el enriquecimiento del texto. Para acceder a esta página basta con pulsar en el botón con la flecha hacia arriba. La Figura 6.3 muestra el estado inicial de esta página.

Como se aprecia en la figura, la página de enriquecido se divide en cuatro partes claramente diferenciadas. La barra superior que se mostraba en la página principal. Una segunda barra superior que muestra un botón flotante con el símbolo “+”, un botón con una flecha hacia abajo para volver al editor y un botón de aplicar en el centro. Aparece una fila con dos “Cajas” donde

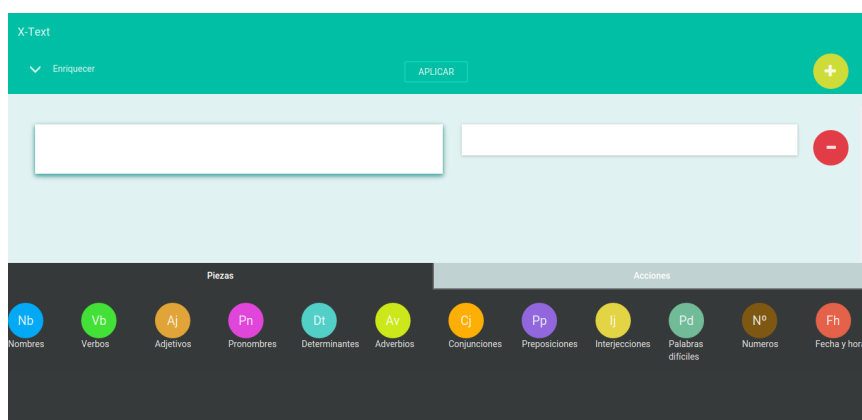


Figura 6.3: Página de enriquecimiento.

se visualizarán las piezas y acciones que se vayan a aplicar o que ya estén aplicadas, junto con un botón con el símbolo “-” para poder eliminar la fila. Por último, en la parte inferior se muestra un panel con dos pestañas en las que encontramos las ya mencionadas piezas y acciones que ofrece el sistema.

A continuación se detallan todos estos elementos y herramientas así como la función que desempeñan.

6.1.3. Panel de piezas y acciones

Este panel consta de dos pestañas en las que, al pulsar, se muestran las herramientas asociadas a dicha pestaña.

La pestaña “Piezas” muestra las piezas que se han considerado en el sistema. Estas piezas identifican las palabras que el usuario quiere reconocer en el texto que previamente introdujo en el editor. El conjunto de piezas que se ofrecen en el sistema se presentan mediante la Tabla 6.1.

Nombres	Preposiciones
Verbos	Interjecciones
Adjetivos	Palabras difíciles
Pronombres	Números
Determinantes	Fecha y Hora
Adverbios	Conjunciones

Tabla 6.1: Piezas ofrecidas por el sistema

La Figura 6.4 muestra todas estas piezas que se acaban de comentar.

Todas estas piezas se aplican siempre a todo el texto que esté en el editor, es decir, si por ejemplo tenemos el texto “El perro de San Roque no tiene

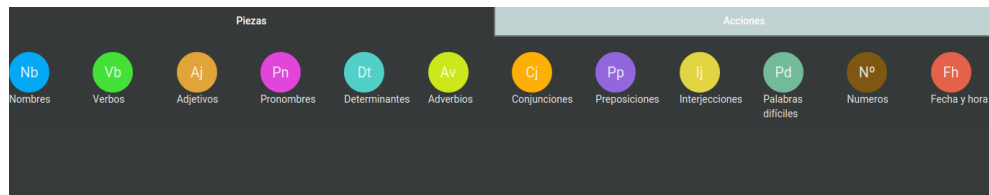


Figura 6.4: Pestaña Piezas.

rabó” y queremos aplicar la pieza Nombres, esta pieza identificará todos los nombres del texto (perro, San Roque y rabo) y se harán transformaciones sobre todos ellos.

Hay que destacar que la pieza “Palabras difíciles” es un poco especial, porque es la única que no está representada dentro de las etiquetas EAGLES. Este tipo de pieza la hemos creado nosotros para añadir funcionalidad a la aplicación.

La pestaña “Acciones” muestra las acciones que se han considerado en el sistema. La Figura 6.5 muestra esta pestaña con todas estas acciones que se presentan a continuación.



Figura 6.5: Pestaña Acciones.

Estas acciones son las herramientas que ofrece el sistema para enriquecer el texto, es decir, son las funciones que realizan las modificaciones sobre las palabras (piezas) que se desean identificar en el texto y las que añaden nueva información al texto para comprender mejor esas palabras. El conjunto de acciones que se ofrece es el siguiente:

La acción de *Subrayar* subraya las palabras asociadas a las piezas que se desean identificar en el texto. La Figura 6.6 muestra un ejemplo de aplicar esta acción a la pieza Nombres y el texto del ejemplo anterior.

La acción de *Análisis Morfológico* analiza morfológicamente las palabras a identificar en texto y añade esta información al final del texto. La Figura 6.7 muestra un ejemplo de aplicar esta acción.

La acción de *Sinónimos* añade una lista de sinónimos al final del texto de las palabras a identificar. La Figura 6.8 muestra un ejemplo de aplicar esta acción.

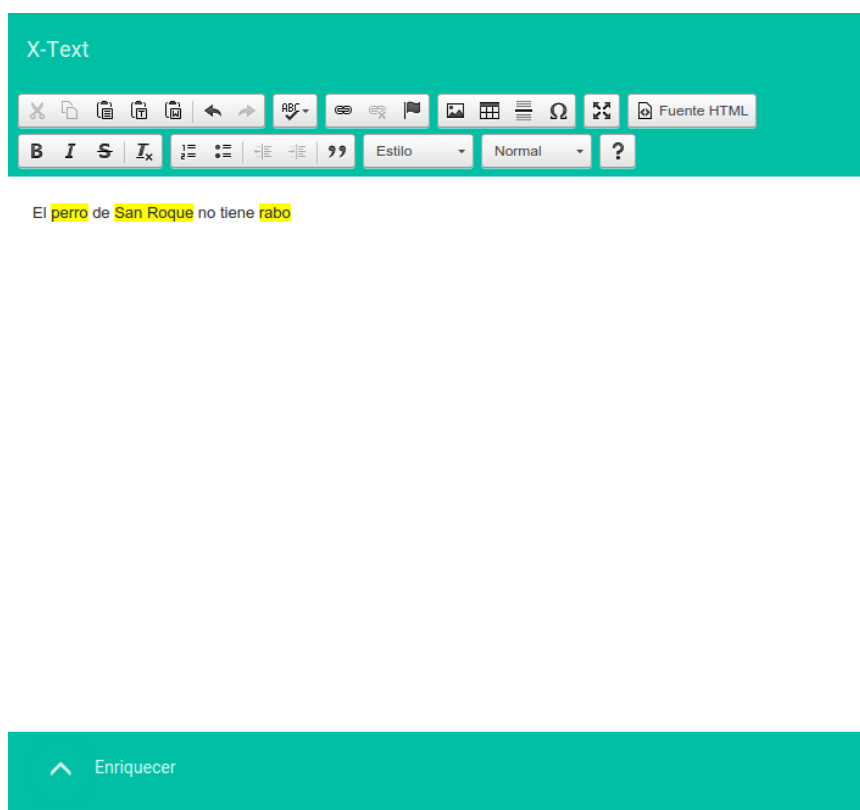


Figura 6.6: Resultado de aplicar la acción Subrayar.

La acción de *Antónimos* añade una lista de antónimos al final del texto de las palabras a identificar. Dado que el ejemplo con el que estamos ilustrando estas acciones no contiene palabras con antónimos, esta vez utilizaremos como ejemplo el texto “Raquel y Susi son dos personas muy honestas y sinceras” y la pieza Adjetivos. La Figura 6.9 muestra este ejemplo.

La acción de *Definiciones* añade al final del texto las definiciones de las palabras que se desean identificar. La Figura 6.10 muestra un ejemplo de aplicar esta acción.

La acción de *Traducción a inglés* traduce al inglés las palabras que se desean identificar en el texto y añade esta información al final. La Figura 6.11 muestra un ejemplo de aplicar esta acción.

Estas acciones pueden combinarse entre ellas de todas las formas posibles para obtener distintos resultados. Hay que tener en cuenta que el orden en el que estén las acciones en una fila no importa. Esto es, si por ejemplo queremos subrayar los nombres que hay en el texto del ejemplo anterior y queremos que se analicen morfológicamente nos dará el resultado de la Figura 6.12.

Este resultado será el mismo si hacemos primero el análisis morfológico

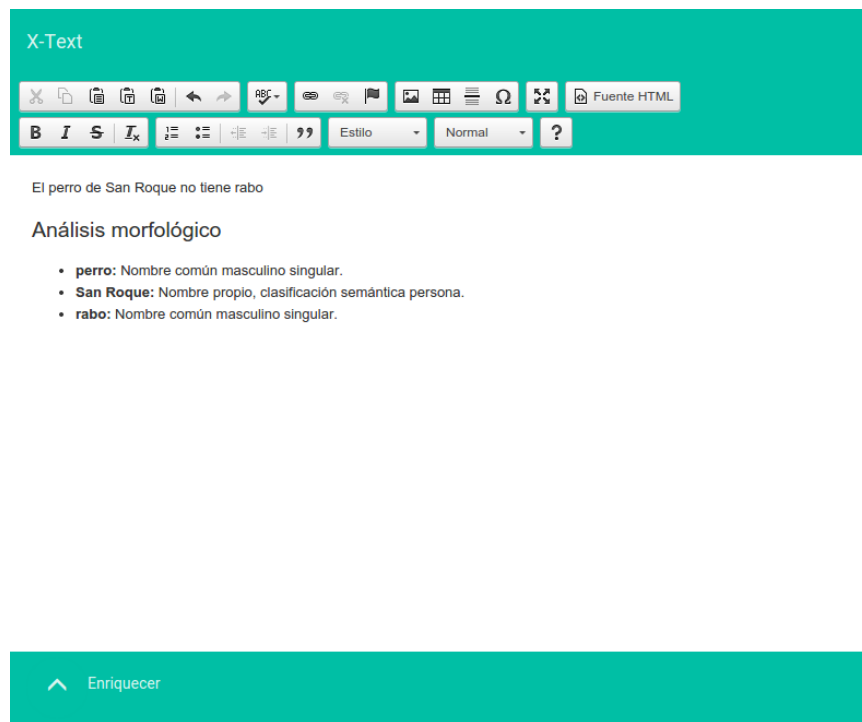


Figura 6.7: Resultado de aplicar la acción Análisis Morfológico.

y después el subrayado de los nombres.

La forma de interactuar con todas estas piezas y acciones es muy simple, basta con pulsar en los botones redondos asociados a las piezas y acciones que se deseen y se añadirán (en caso de no estar añadidas) o eliminarán (en caso de estar añadidas) a las cajas que están arriba y habrá que pulsar el botón aplicar para que se produzcan los cambios en el texto. Además, pueden eliminarse piezas y acciones de las cajas pulsando sobre ellas en lugar de usar el panel inferior.

6.1.4. Cajas de piezas y acciones

Como se ha comentado, la utilidad de estos elementos reside únicamente en la visualización de las piezas y acciones que se van a aplicar o ya se han aplicado. Estas “Cajas” son modificables, es decir, el contenido de estas puede modificarse después de haber sido aplicadas añadiendo otras piezas y acciones o eliminando las que ya están.

Además, se pueden añadir más filas pulsando el botón flotante con el símbolo “+” y se añadirán dos cajas por cada fila, una para piezas y otra para acciones. Esto hace que se puedan combinar varias filas sin que tengan nada que ver las piezas y acciones que tienen cada una de ellas. El texto

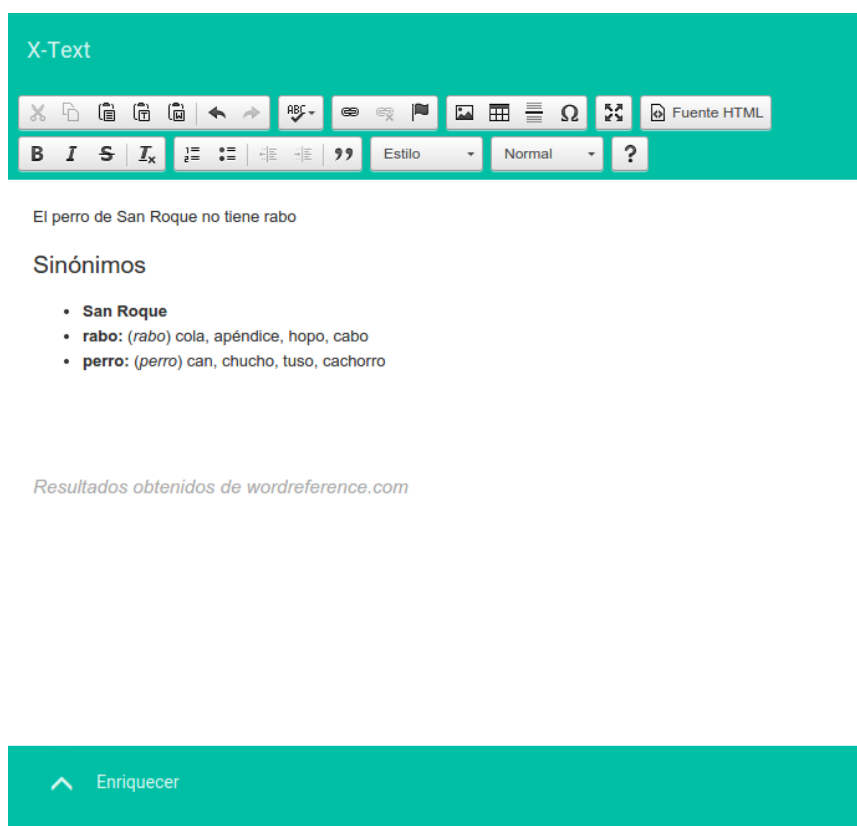


Figura 6.8: Resultado de aplicar la acción Sinónimos.

resultante será la combinación de todas ellas.

La Figura 6.13 muestra el resultado de aplicar dos filas, una con subrayar nombres y otra con análisis morfológico de los verbos al texto que estamos utilizando de ejemplo.

Para eliminar una fila simplemente hay que pulsar sobre el botón con el símbolo “-” asociado a la fila y ésta desaparecerá.

6.1.5. Botón aplicar

Como se ha ido mencionando hasta ahora, para establecer los cambios que se desean sobre el texto hay que pulsar el botón “Aplicar”. Para poder aplicar una o una serie de filas al texto, éstas deben contener al menos una pieza y una acción por fila, en caso de no haber ninguna pieza o acción en una fila o que alguna de las “Cajas” de una fila esté vacía, dichas “Cajas” se muestran con el fondo de color rojo para indicar al usuario que debe poner alguna pieza o acción como ilustra la Figura 6.14.

Además, este botón es el que implementa la conexión con el servicio web

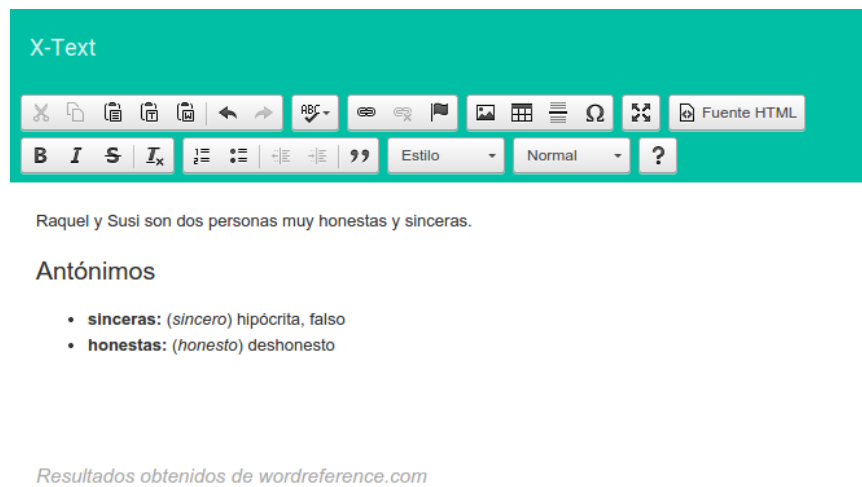


Figura 6.9: Resultado de aplicar la acción Antónimos.

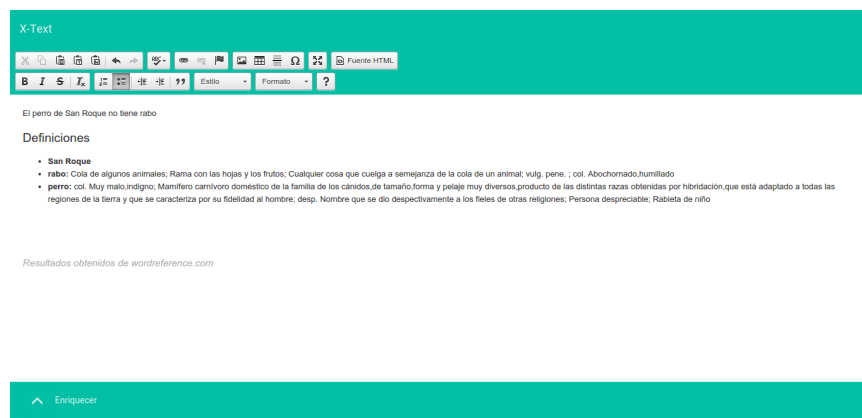


Figura 6.10: Resultado de aplicar la acción Definiciones.

alojado en el servidor que se cuenta en el siguiente capítulo.

La conexión se ha implementado mediante la utilización de la herramienta

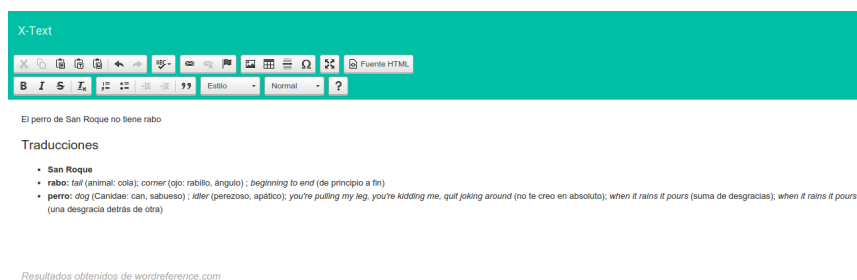


Figura 6.11: Resultado de aplicar la acción Traducción a inglés.

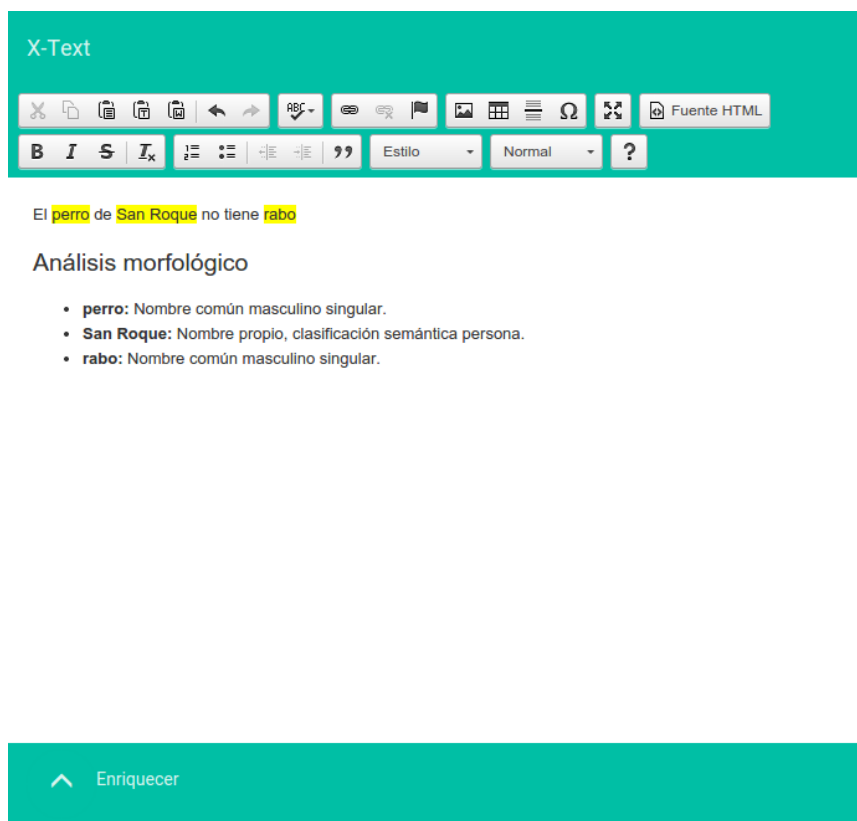


Figura 6.12: Resultado de aplicar las acciones Subrayar y Análisis Morfológico.

AJAX. Como se comentó en el capítulo de Estado del Arte, AJAX es una función para hacer peticiones HTTP a un servicio web que implemente este

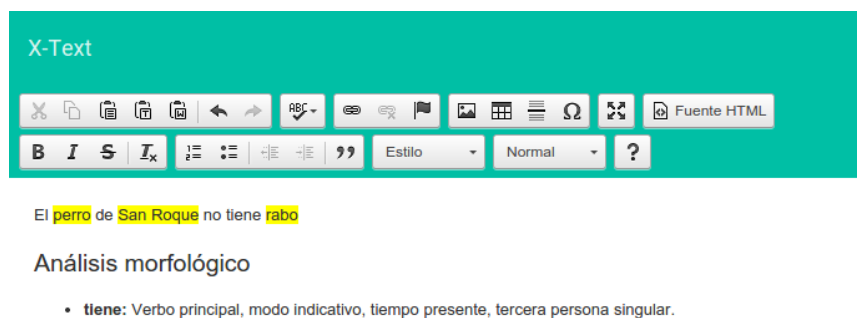


Figura 6.13: Resultado de aplicar dos filas.

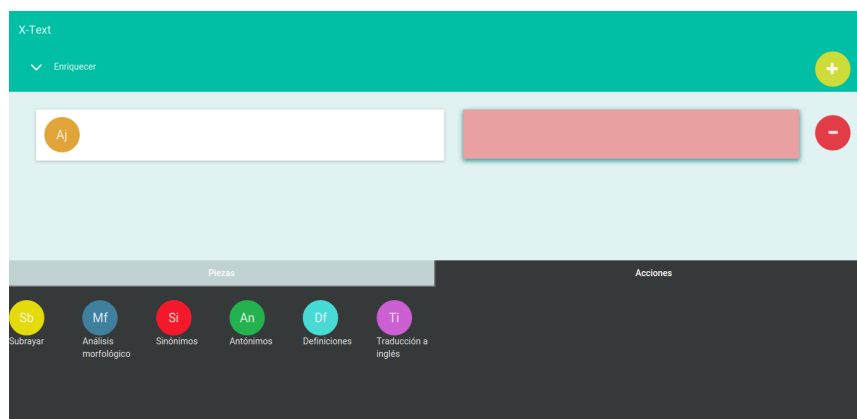


Figura 6.14: Caja de acciones vacía.

protocolo.

Este botón realiza una petición con todas las filas que contengan piezas y acciones, el texto que está actualmente en el editor y otros parámetros

que viajan en formato JSON y que se detallan en el siguiente capítulo. Esta petición se realiza de forma asíncrona, es decir, la petición se hace en segundo plano lo que permite a la aplicación seguir con su ejecución sin problemas.

Por último, cuando se pulsa este botón aparece un diálogo modal para que el usuario sepa que se está procesando su petición como muestra la Figura 6.15.

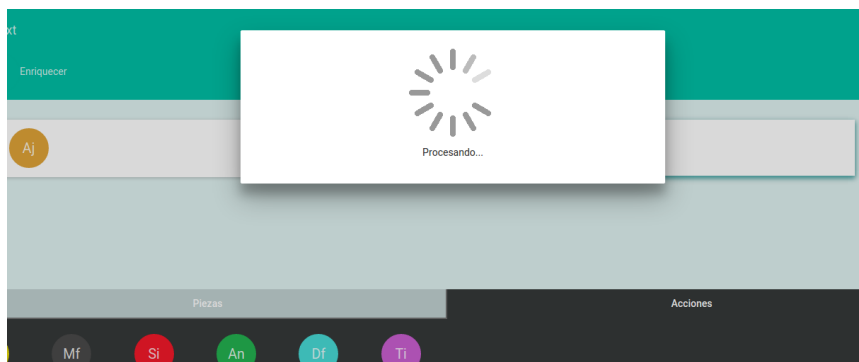


Figura 6.15: Diálogo modal.

6.1.6. Discusión sobre las evaluaciones iniciales

Como se puede apreciar, no se han podido implementar todas las funcionalidades que se pensaron en la construcción del boceto así como algunas de las formas de interactuar con la interfaz. Esto ha ocurrido debido al tiempo limitado del proyecto y a todas las modificaciones de posibles funcionalidades que se implementaron y que no han resultado útiles e intuitivas a la hora de utilizar la aplicación como se cuenta en el capítulo 8 con las evaluaciones heurísticas.

Las funcionalidades que no se han llegado a implementar son:

- El panel lateral izquierdo que contenía las funciones para cargar o guardar texto en diferentes formatos así como poder guardar el texto enriquecido con las piezas y acciones aplicadas, el gestor de piezas y acciones para poder crear, modificar y eliminar piezas y acciones, y ayuda por si el usuario tuviera dificultades a la hora de utilizar la aplicación.
- El panel lateral derecho que contenía el historial en el cual se mostraban los textos enriquecidos con anterioridad junto con su fecha y hora de la última modificación.

En cuanto a las formas de interactuar que se definieron en el boceto, se ha cambiado la forma de eliminar las filas que en un principio se hacía

deslizando la fila hacia la izquierda, por un botón con el símbolo “-” debido a que esa forma de interactuar no nos resultó intuitiva cuando se utiliza la aplicación en un ordenador que no es táctil. Adicionalmente se ha añadido la posibilidad para eliminar piezas y acciones pulsando sobre ellas en la “Cajas” ya que resulta mucho más intuitivo y mejora la experiencia de usuario.

Capítulo 7

Servidor

Como acabamos de ver en el capítulo anterior, la interfaz web es la que se encarga de llamar al servicio principal de la aplicación que es la que se encarga de enriquecer el texto. Este servicio principal es el encargado de gestionar todo lo que necesite la parte web y de devolver el texto enriquecido con un formato adecuado. Para esto, usamos ciertos servicios web como el de Freeing que nos ayuda a analizar el texto y sacar toda la información que necesitamos para enriquecerlo posteriormente. Sin embargo, la información que proporcionan estos servicios necesitan un tratamiento e integración en el texto para ser realmente útiles. Este procesamiento se realiza en un servidor que almacena toda la información necesaria para realizar las funcionalidades de transformación de texto de las que dispone nuestra aplicación.

En las próximas secciones hablaremos con más detalle sobre como actúa el servicio principal, como se llaman a los servicios que enriquecen el texto, la información que devuelven, y cómo se usa esta información para enriquecer el texto.

7.1. Servicio principal

El servicio principal es el encargado de recibir las peticiones de la parte web y de generar toda una serie de llamadas a servicios y procesamientos que generan como resultado un texto enriquecido personalizado como quiera el usuario del sistema. Como ya hemos visto, estas peticiones se realizan a través de un objeto JSON que se le envía al servicio y que tiene el siguiente formato:

```
{texto: <Texto a enriquecer>, filas: [piezas: <Piezas1>, acciones: <Acciones1>, id: <Id1>}, ... , {piezas: <Piezasn>, acciones: <Accionesn>, id: <Idn>}]}
```

Un JSON no es más que un objeto con atributos que tienen unos valores. En nuestro caso este objeto tiene los atributos texto, piezas, acciones e id.

Y por tanto sus valores son `<Texto a enriquecer>`, `<Piezas>`, `<Acciones>` e `<Id>`.

Un JSON no es más que un objeto con atributos que tienen unos valores. En nuestro caso este objeto tiene los atributos texto y filas.

El texto puede tener como valor `<Texto a enriquecer>`, y el atributo filas es un array de “n” elementos JSON que tiene como atributos las piezas, acciones e id. Los valores que pueden tener estos últimos atributos son `<Piezas1-n>`, `<Acciones1-n>` e `<Id1-n>`. .

Los atributos tienen que estar en un formato que hemos prefijado nosotros. A continuación, iremos viendo uno a uno todos los atributos y los explicaremos con más detalle:

- **texto:** En este campo hay que poner el texto que queremos enriquecer. Este texto no tiene por qué ser texto plano, si no que podemos introducir texto con el formato que tendría un archivo HTML. Esto es de gran utilidad para nosotros ya que el texto que el usuario introduce en el editor CKEditor, es formateado directamente por él. Es decir, introduce las etiquetas HTML necesarias para poner el texto con un formato más atractivo para el usuario. Incluso el usuario, al usar las herramientas del editor, también hace que se introduzcan ciertas etiquetas. Por ejemplo, para crear un título con una cabecera especial, el usuario puede directamente seleccionar el texto que va a ser el título, irse al botón del formato del párrafo y seleccionar cualquiera de los tipos “Heading”. Lo que hace CKEditor es introducir una etiqueta `<h1>...</h1>` que engloba todo el título, y entonces el usuario puede ver el título más grande. Además puede verse como formatea el texto CKEditor si dentro del editor pulsamos en el botón “Fuente HTML”, e incluso el usuario podría editar el texto modificando directamente el texto fuente HTML.
- **filas:** En este campo hay que poner un array de todas las filas que se vayan a ejecutar. Estas filas, como hemos adelantado previamente, son JSON que tienen su propio formato, cuyos atributos se van a explicar en los siguientes puntos.
- **piezas:** En este campo hay que poner las piezas que queremos usar. Como ya hemos hablado en el capítulo anterior, estas piezas serán a las que se le apliquen las acciones que se encuentren en la misma fila. Para poner estas piezas es muy importante seguir el formato que se presenta en la Tabla 7.1 y que hemos definido nosotros mismos. Salvo las piezas difíciles, todas las piezas siguen el mismo formato que se usa para las etiquetas EAGLES. Si lo que queremos es poner varias piezas, separaremos las piezas por dos puntos. Por ejemplo, si queremos usar las piezas “Nombre” y “Adjetivo” tendremos que poner en el JSON “piezas:n:a” (también valdría “piezas:n:a:”).

<Pieza>	Formato
Adjetivo	a
Adverbio	r
Determinante	d
Nombre	n
Verbo	v
Pronombre	p
Conjunción	c
Interjección	i
Preposicion	s
Numeral	z
Fecha/Hora	w
Difícil	difícil

Tabla 7.1: Formato de las piezas en el objeto JSON

- **acciones:** En este campo hay que poner las acciones que queremos usar. Como ya hemos hablado en el capítulo anterior, estas acciones serán las que se apliquen a las piezas que se encuentren en la misma fila. Para poner estas piezas es muy importante seguir el formato que se presenta en la siguiente Tabla 7.2: Si lo que queremos es poner va-

<Acción>	Formato
Subrayar	subrayar
Análisis morfológico	morfo
Buscar sinónimos	sinonimos
Buscar antónimos	antonimos
Buscar definiciones	definiciones
Buscar traducciones	traducciones

Tabla 7.2: Formato de las piezas en el objeto JSON

rias acciones, separaremos las acciones por dos puntos. Por ejemplo, si queremos usar las acciones “Subrayar” y “Análisis morfológico” tendremos que poner en el JSON “acciones:subrayar:morfo” (también valdría “acciones:subrayar:morfo:”).

- **id:** En este campo hay que poner una clave que identifique inequívocamente al usuario. Nosotros usamos el método `getTime()` de la clase “Date” de JavaScript, que devuelve el número de milisegundos desde 1970/01/01, ya que es muy difícil que dos usuarios realicen una acción en el mismo milisegundo. Para asegurar el correcto funcionamiento de la aplicación, aconsejamos usar la misma metodología cuando se use este atributo o cualquier otra metodología que sea inequívoca, como nom-

bre+apellidos+fecha+hora. No podremos asegurar su funcionamiento si se usase otro tipo de metodología. Esta clave ahora mismo no se usa, pero se podría usar para guardar todas las acciones que se hacen en la aplicación y así poder sacar estadísticas de uso y otras funciones de las que hablaremos en los capítulos 9 y 10.

Ahora que ya sabemos que es lo que recibe el servicio principal, veremos como se llama a este servicio.

7.2. Llamada al servicio principal

La manera de acceder al servicio principal es mediante una petición HTTP POST. Esta petición debe llevar como datos el objeto JSON mencionado en el apartado anterior. Por lo tanto, para acceder a este servicio necesitamos tener la URL del servicio principal y el objeto JSON que deseemos.

La dirección URL del servicio principal es la siguiente:

`http://sesat.fdi.ucm.es:8080/xtextws/rest/XtextWS/chooseService`

Para entender mejor como se realiza esta petición utilizando la herramienta AJAX desde una página web, rescataremos el ejemplo del capítulo anterior en el que el texto era “El perro de San Roque no tiene rabo” y aplicando las piezas Nombres, Verbos y las acciones Subrayar, Sinónimos y Análisis morfológico.

El objeto JSON que se tendría que enviar sería de la siguiente manera:

`{texto: “El perro de San Roque no tiene rabo”, filas: [{ piezas: “n:v”, acciones: “subrayar:sinonimos:morfo”, id: “15646143453” }]}`

Los parámetros de la petición AJAX serían de los siguientes:

type: “POST”

url: “http://sesat.fdi.ucm.es:8080/xtextws/rest/XtextWS/chooseService”

contentType: “application/json; charset=UTF-8”

data: `JSON.stringify(texto: “El perro de San Roque no tiene rabo”, filas: [{ piezas: “n:v”, acciones: “subrayar:sinonimos:morfo”, id: “15646143453” }])`

La Figura 7.1 ilustra este ejemplo con más detalle.

7.3. Funcionamiento del servicio principal

Ya hemos adelantado en capítulos anteriores qué funciones tiene nuestro sistema, y también hemos hablado de cómo se llama al servicio y lo que

```
var json = {texto: "El perro de San Roque no tiene rabo", [piezas: "n:v", acciones: "subrayar:norfo", id: 15646143453]};  
var stringjson = JSON.stringify(json);  
  
$.ajax({  
  type: "POST",  
  url: "http://sesat.fdi.ucm.es:8080/xtxtws/rest/XtxtWS/chooseService",  
  contentType: "application/json; charset=UTF-8",  
  data: stringjson,  
  success: function(respuesta) {  
    alert(respuesta);  
  },  
  error: function(e) {  
    alert("Error: " + e);  
  }  
});
```

Figura 7.1: Código JavaScript de la petición AJAX

recibe. Ahora hablaremos de cómo realiza el servicio principal esas funciones más detalladamente.

El proceso de análisis y transformación de un texto, tal como lo realizamos en nuestro sistema, es una tarea bastante compleja y en la que intervienen diversos componentes. Para explicar este proceso de manera que se entienda lo mejor posible, dividiremos este proceso en varias etapas que explicaremos en las siguientes subsecciones.

Estas etapas son:

1. Inicializar ejecución
2. Eliminar enriquecido
3. Tokenización del texto
4. Unir acciones y tokens
5. Aplicar la ejecución
6. Guardar y mostrar

7.4. Etapas del servicio principal

7.4.1. Inicializar ejecución

Como ya veremos en la siguiente etapa, la tarea de analizar todo el texto es una tarea muy difícil y la más compleja computacionalmente del sistema, por lo que es la que más tiempo tarda en ejecutarse y la que más convendría reducir en un trabajo futuro.

Para prepararnos para esa etapa, lo que haremos es guardar primero toda la información que podamos acerca de la ejecución que se está aplicando en cierto instante. Para ello, parsearemos el objeto JSON inicial y sacaremos la siguiente información:

- Piezas que se van a ejecutar: Guardaremos todos los tipos de tokens que se van a ejecutar en una tabla hash para poderlos consultar rápidamente. Esto nos va a venir bien, porque si sabemos esta información y sabemos qué acciones se van a aplicar a un tipo de pieza, lo tenemos todo para aplicar la ejecución.
- Acciones que se van a ejecutar: Guardaremos todos los tipos de acciones que se van a ejecutar en una tabla hash para poderlos consultar rápidamente. Esto nos va a venir bien, porque dependiendo de qué acciones se vayan a aplicar la aplicación reaccionará de distintos modos, tal como se explicará en las siguientes etapas.
- Piezas por acción: Guardaremos todos los tipos de tokens que se van a ejecutar, y para qué acciones. Para ello usaremos una tabla hash para cada acción, para poder consultar esta información rápidamente. Esto nos valdrá para saber qué tokens tenemos que ejecutar para cada acción cuando las lancemos en etapas posteriores.

7.4.2. Eliminar enriquecido

Como ya hemos visto en el capítulo 6, el sistema aplica las filas sobre el texto original sin enriquecer. Por lo tanto, si ya se hubiese aplicado una ejecución, todo el enriquecimiento del texto tendría que ser eliminado antes de aplicar otra ejecución distinta.

Para ello, el trabajo a realizar para eliminar el texto enriquecido dependerá del tipo de acción que se haya llevado a cabo:

- Subrayado: Para esta acción, lo que hará la aplicación será eliminar todas las palabras subrayadas del texto. Para ello recorreremos todo el texto y eliminaremos las etiquetas de subrayado para todos los fragmentos de texto que estén subrayados.

Esto implica que si el usuario ha realizado una acción de subrayado sobre el texto manualmente, sin aplicar enriquecimiento, también se eliminará tras aplicar una nueva ejecución.

- Análisis morfológico: Para esta acción, lo que hará la aplicación será eliminar todos los fragmentos que tengan el formato de un texto enriquecido con un análisis morfológico. Es decir, eliminará todas aquellas cabeceras de la forma “<h2>Análisis Morfológico </h2>” que posteriormente contenga una lista de elementos de la forma “...”.

Esto implica que si el usuario ha escrito manualmente el mismo formato que tiene el análisis morfológico, o ha copiado y pegado la parte del

análisis morfológico de un texto enriquecido en otro lugar, también se eliminará tras aplicar una nueva ejecución.

- Antónimos, sinónimos, definiciones y traducciones: Para estas acciones se realizará el mismo procedimiento que para el análisis morfológico, con la diferencia de que se eliminarán aquellas cabeceras que contengan el título de la correspondiente acción. Además, otra diferencia que tienen estas acciones con la del análisis morfológico, es que también hay que borrar la referencia que indica que esos resultados han sido sacados de la web de WordReference.

Todo esto implica que si el usuario ha puesto en el texto manualmente cualquiera de estos formatos, serán eliminados tras aplicar una nueva ejecución.

7.4.3. Tokenización del texto

Ya hemos adelantado que esta etapa es la más compleja computacionalmente. Ahora vamos a ver el por qué.

La tokenización del texto se realiza casi en su totalidad con el servicio web de Freeling. Como ya hemos contado en el capítulo del procesamiento del lenguaje natural, el servicio de Freeling analiza todo el texto y nos devuelve todas las palabras del texto con su representación con etiquetas EAGLE, así como el índice de comienzo y fin de la palabra en el texto.

Llamar a este servicio tarda la mayor parte del tiempo total de ejecución y además no nos da la información de las palabras difíciles. Además de ese problema, hay que añadir otro más: el servicio web de Freeling no entiende la codificación HTML para caracteres especiales como las tildes, las eñes, las diéresis, y otros caracteres como “o”, “a”, “i”, “¿”, “»” y el carácter de espacio.

Además, el texto que nos devuelve la parte web del sistema, hemos visto anteriormente que CKEditor lo formatea añadiendo las etiquetas HTML que hagan falta. Si le enviásemos a Freeling el texto con todas etiquetas, perdería todo el contexto del texto, ya que una etiqueta del tipo “<p>”, Freeling lo entendería como “<:signo de puntuación, p=nombre, >:signo de puntuación”, por lo que también tenemos que hacer algo para arreglar esto.

Como tenemos varios problemas muy importantes, lo que hacemos es realizar una primera pasada al texto antes de enviarlo a Freeling, en la cual cambiamos los caracteres HTML especiales por sus correspondientes caracteres en codificación UTF8; analizamos qué palabras son difíciles; y quitamos las etiquetas HTML. En este punto de la ejecución también realizaremos la búsqueda de las piezas que se necesitan en la ejecución actual, pero esto lo contaremos más detalladamente en la siguiente etapa.

Por otro lado, y aprovechando que hay que recorrer el texto dos veces para analizar todo el texto, y que es una tarea muy compleja computacionalmente, crearemos objetos de todas las palabras con toda la información que saquemos de la tokenización, para guardarlos y que no tengamos que volver a tokenizar para la ejecución que estemos realizando.

Otra labor que se realiza ahora es ver cuáles de los tokens son palabras difíciles. Esto se contará en la Sección 7.4.3.1

Una vez realizado este parseo ya podremos enviarle el texto a Freeling.

Para usar este servicio, hemos optado por importarlo a nuestro código con la herramienta JAX-WS, de la cual hemos hablado en el capítulo de estado del arte 3.2.5. Esto nos generaría dentro de nuestro proyecto una clase “Freeling3TaggingClient” que tiene un método el siguiente método “runOutput”:

```
public static String runOutput(  
    //@WebParam(name = "input_direct_data", targetNamespace = "")  
    String inputDirectData,  
    //@WebParam(name = "input_url", targetNamespace = "")  
    String inputUrl,  
    //@WebParam(name = "language", targetNamespace = "")  
    String language,  
    //@WebParam(name = "keeptags", targetNamespace = "")  
    Boolean keeptags,  
    //@WebParam(name = "noner", targetNamespace = "")  
    Boolean noner,  
    //@WebParam(name = "bio", targetNamespace = "")  
    Boolean bio,  
    //@WebParam(name = "nonec", targetNamespace = "")  
    Boolean nonec,  
    //@WebParam(name = "flush", targetNamespace = "")  
    Boolean flush,  
    //@WebParam(name = "noafx", targetNamespace = "")  
    Boolean noafx,  
    //@WebParam(name = "noloc", targetNamespace = "")  
    Boolean noloc,  
    //@WebParam(name = "nonumb", targetNamespace = "")  
    Boolean nonumb,  
    //@WebParam(name = "nopunt", targetNamespace = "")  
    Boolean nopunt,  
    //@WebParam(name = "nodate", targetNamespace = "")  
    Boolean nodate,  
    //@WebParam(name = "noquant", targetNamespace = "")  
    Boolean noquant,
```

```
//@WebParam(name = "nodict", targetNamespace = "")
Boolean nodict,
//@WebParam(name = "nopro", targetNamespace = "")
Boolean nopro,
//@WebParam(name = "xmlcqp", targetNamespace = "")
Boolean xmlcqp){...}
```

Esta función, si nos fijamos, tiene los mismos atributos que tiene la interfaz web del servicio (Figura 7.2):

De esta función, los parámetros que nos interesan y que vamos a explicar son: “inputDirectData”, “inputUrl” y “language”. “inputDirectData” es el texto que se le va a enviar directamente a Freeling. Nosotros este parámetro no lo usamos (le ponemos un string vacío, no se le puede poner a null) ya que no admite textos largos. Para solucionar esto lo que hacemos es usar el parámetro “inputUrl”. Lo que hace éste es coger el texto contenido en dicha URL, y esto sí permite coger textos largos.

Sin embargo, cuando la parte web de la aplicación manda al servidor el texto, no tenemos en ningún momento ninguna URL que tenga el texto. Para arreglar esto, lo que hacemos es hacer público el texto, guardando el texto en un archivo en el servidor. Para hacerlo público hay un directorio en el servidor que es accesible por cualquier IP que es “/var/www/html”. Este es el directorio raíz de la web www.sesat.fdi.ucm.es. Todo lo que se ponga aquí, incluidos los directorios se harán públicos y accesibles a través de la dirección www.sesat.fdi.ucm.es/<Dirección del recurso>.

El último parámetro que nos interesa es “language”. Este parámetro representa en lenguaje en el que está escrito el texto. En nuestro caso será español, identificado por “es”.

De este modo, cuando llamemos al servicio de Freeling, nos dará como salida la misma que nos daría la versión web de Freeling (Figura 7.3). Para saber más acerca del formato de la salida de Freeling se puede consultar el capítulo 3.1.1.

Una vez que tengamos la salida, podemos parsearla y guardar en los objetos de los token toda la información que hemos obtenido.

7.4.3.1. Pieza Palabras Dificiles

Es importante mencionar que esta pieza la hemos creado nosotros ya que no la realiza ningún servicio externo como el de Freeling o el de Maltparser.

En cuanto a la tokenización de este tipo de pieza, hay que tener en cuenta que cualquier token, salvo los signos de puntuación, puede ser, además, una palabra difícil.

Inputs

No file selected.

☐ as URL
☒ direct data or local file

☒ language en

☒ **bio**

yes

no

☒

☒ **flush**

yes

no

☒

☒ **keeptags**

yes

no

☒

☒ **noafx**

yes

no

☒

☒ **nodate**

yes

no

☒

☒ **nodict**

yes

no

☒

☒ **noloc**

yes

no

☒

☒ **nonec**

yes

no

☒

☒ **noner**

yes

no

☒

☒ **nonumb**

yes

no

☒

☒ **noprob**

yes

no

☒

☒ **nopunt**

yes

no

☒

☒ **noquant**

yes

no

☒

☒ **xmlcqp**

yes

no

☒

mandatory

optional

[Reset fields](#)

Figura 7.2: Versión web de Freeling

Para saber si una palabra es difícil, lo que tenemos es un archivo que contiene las 10.000 palabras más usadas del castellano. De este modo, suponemos que si una palabra no está entre las 10.000 palabras más usadas

esto	este	PD0NS000	1	0	4
es	ser	VSIP3S0 1	5	7	
una	uno	DI0FS0 0.951241		8	11
prueba	prueba	NCFS000 0.916667		12	18

Figura 7.3: Ejemplo de salida de Freeling

del castellano, es porque es una palabra difícil. Como en este archivo no se encuentran los signos de puntuación, y como no queremos que los signos de puntuación sean palabras difíciles, lo que hemos hecho es introducir los signos de puntuación en el archivo. De este modo tendríamos en el archivo las 10.000 palabras más usadas del castellano y los signos de puntuación.

7.4.4. Unir acciones y tokens

En la fase anterior hemos tokenizado todo el texto y hemos extraído los tokens que se van a ejecutar, pero todavía no sabemos qué acciones hay que aplicar a todos esos tokens. Sin embargo, gracias a la fase de inicialización sabemos qué acciones hay que aplicar a todos los tipos de tokens.

Lo que haremos será recorrer todos los tokens que hemos extraído del texto, y los introduciremos en tablas diferentes dependiendo de la acción. Estas tablas son tablas hash en las que guardamos la piezas con una clave que es el correspondiente texto de la pieza. De este modo, tendremos una serie de tablas para cada acción, con todas las piezas que hay que aplicar.

7.4.5. Aplicar la ejecución

Para saber qué piezas vamos a tener que tratar, solo tenemos que comparar en la salida de Freeling para ver si la etiqueta EAGLES coincide con una de las acciones que se encuentran en el objeto JSON que hemos explicado al principio de este capítulo. De este modo, si tenemos las piezas “nombre” y “verbo”, cogeríamos todas las piezas cuya etiqueta EAGLE incluya en su letra inicial el formato de un nombre (es decir, “n”) o el formato de un verbo (es decir, “v”).

Ahora que ya sabemos qué acciones hay que aplicar, y para qué piezas, solo tenemos que ejecutar esas acciones. Según el tipo de acción que sea, la ejecución se realizará de la siguiente manera:

- **Subrayar:** Para esta acción tendremos que poner una etiqueta HTML de la forma “texto”, donde el texto sería la pieza que queremos subrayar. De este modo, cuando se lo pasemos a CKEditor, mostrará todas estas piezas subrayadas.

- **Análisis morfológico:** Para esta acción lo único que tendremos que hacer es coger la etiqueta EAGLES de la pieza, la cual hemos guardado en el proceso de parseo de la salida de Freeling, y parsearla para sacar toda la información del análisis morfológico.
- **Buscar sinónimos:** Para esta acción usaremos la web de WordReference (Kellogg, 1999) para conseguir la información que necesitamos. Esta página está escrita en HTML por lo que deberemos recorrerla para sacar solamente la información que queremos, que son las palabras que son sinónimos de las piezas que estamos ejecutando.

Para acceder a esta página introducimos la URL `http://www.wordreference.com/sinonimos/<PalabraABuscar>` donde `<PalabraABuscar>` sería la pieza que estamos ejecutando, y la parseamos para quitar toda la información irrelevante de la página HTML.

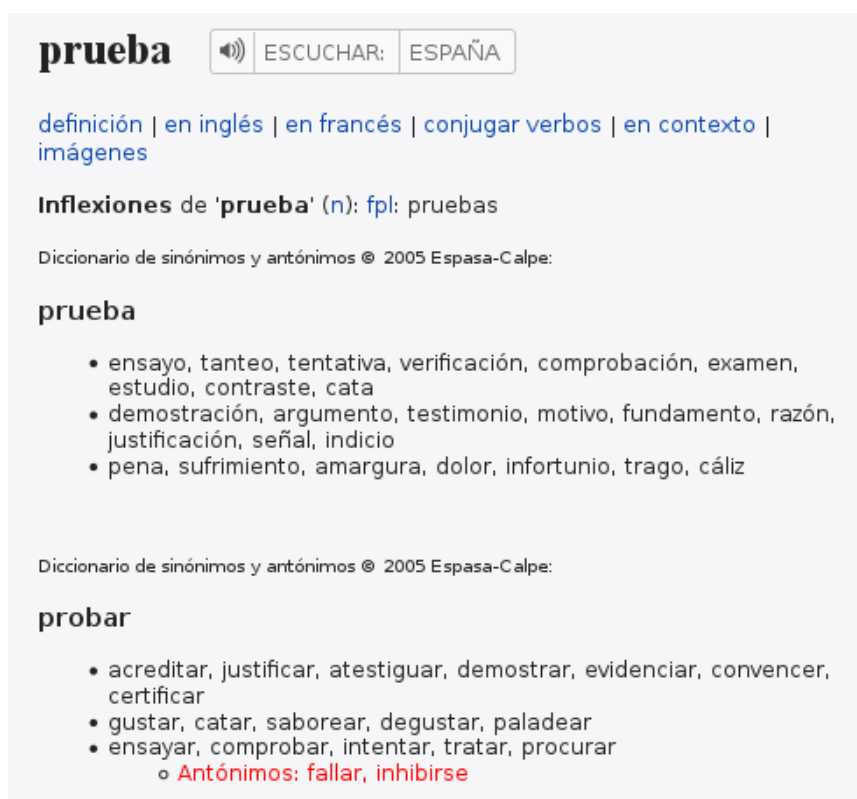
- **Buscar antónimos:** Para esta acción usaremos la web de WordReference (Kellogg, 1999) para conseguir la información que necesitamos. Esta página también está escrita en HTML por lo que deberemos recorrerla para sacar solamente la información que queremos, que son las palabras que son antónimos de las piezas que estamos ejecutando.

Para acceder a esta página solo introducimos la URL `http://www.wordreference.com/sinonimos/<PalabraABuscar>` donde `<PalabraABuscar>` sería la pieza que estamos ejecutando, y la parseamos para quitar toda la información irrelevante de la página HTML. Podemos ver que la URL es la misma que la de sinónimos. Esto se debe a que WordReference usa el mismo servicio para definir tanto los sinónimos como los antónimos. Puedes ver un ejemplo de este servicio en la Figura 7.4.

- **Buscar definiciones:** Para esta acción usaremos la web de WordReference (Kellogg, 1999) para conseguir la información que necesitamos. Esta página también está escrita en HTML por lo que deberemos recorrerla para sacar solamente la información que queremos, que son las palabras que son definiciones de las piezas que estamos ejecutando.

Para acceder a esta página solo introducimos la URL `http://www.wordreference.com/definicion/<PalabraABuscar>` donde `<PalabraABuscar>` sería la pieza que estamos ejecutando, y la parseamos para quitar toda la información irrelevante de la página HTML. Puedes ver un ejemplo de este servicio en la imagen 7.5

- **Buscar traducciones:** Para esta acción usaremos la web de WordReference (Kellogg, 1999) para conseguir la información que necesitamos.



The screenshot shows the wordreference.com interface for the word 'prueba'. At the top, the word 'prueba' is displayed in a large, bold font. To its right is a speaker icon and a button labeled 'ESCUCHAR: ESPAÑA'. Below this, there are links for 'definición', 'en inglés', 'en francés', 'conjugar verbos', 'en contexto', and 'imágenes'. The main section is titled 'Inflexiones de 'prueba' (n): fpl: pruebas'. Below this, it says 'Diccionario de sinónimos y antónimos © 2005 Espasa-Calpe:'. The word 'prueba' is listed again, followed by a bulleted list of synonyms: 'ensayo, tanteo, tentativa, verificación, comprobación, examen, estudio, contraste, cata', 'demostración, argumento, testimonio, motivo, fundamento, razón, justificación, señal, indicio', and 'pena, sufrimiento, amargura, dolor, infortunio, trago, cáliz'. Below this list, it says 'Diccionario de sinónimos y antónimos © 2005 Espasa-Calpe:'. The word 'probar' is listed, followed by a bulleted list of synonyms: 'acreditar, justificar, atestiguar, demostrar, evidenciar, convencer, certificar', 'gustar, catar, saborear, degustar, paladear', and 'ensayar, comprobar, intentar, tratar, procurar'. Below this list, it says 'Antónimos: fallar, inhibirse'.

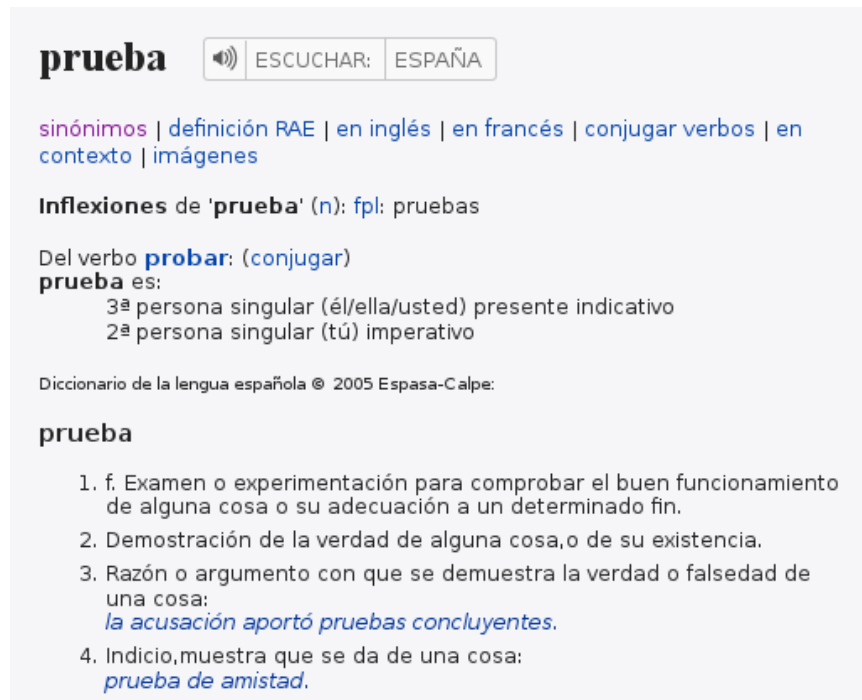
Figura 7.4: Ejemplo de resultados de wordreference para sinónimos y antónimos


Esta página también está escrita en HTML por lo que deberemos recorrerla para sacar solamente la información que queremos, que son las palabras que son las traducciones de las piezas que estamos ejecutando.

Para acceder a esta página introducimos la URL `http://www.wordreference.com/es/en/translation.asp?spen=<PalabraABuscar>` donde `<PalabraABuscar>` sería la pieza que estamos ejecutando, y la parseamos para quitar toda la información irrelevante de la página HTML. Puedes ver un ejemplo de este servicio en la Figura 7.6.

7.4.6. Guardar y mostrar

La ejecución realizada en la etapa de ejecución generará varios textos según cuáles fuesen las acciones a ejecutar. Es decir, en cualquier ejecución habrá un texto para el análisis morfológico, otro para subrayar, otro para los sinónimos, otro para las definiciones y otro distinto para las traducciones. Estos estarán todos inicialmente vacíos salvo el de subrayar, que será el texto que teníamos en el JSON inicialmente sin enriquecer. Los textos que



prueba  ESCUCHAR: ESPAÑA

[sinónimos](#) | [definición RAE](#) | [en inglés](#) | [en francés](#) | [conjugar verbos](#) | [en contexto](#) | [imágenes](#)

Inflexiones de 'prueba' (n): [fpl](#): pruebas

Del verbo **probar**: ([conjugar](#))
prueba es:
 3ª persona singular (él/ella/usted) presente indicativo
 2ª persona singular (tú) imperativo

Diccionario de la lengua española © 2005 Espasa-Calpe:

prueba

1. f. Examen o experimentación para comprobar el buen funcionamiento de alguna cosa o su adecuación a un determinado fin.
2. Demostración de la verdad de alguna cosa, o de su existencia.
3. Razón o argumento con que se demuestra la verdad o falsedad de una cosa:
[la acusación aportó pruebas concluyentes.](#)
4. Indicio, muestra que se da de una cosa:
[prueba de amistad.](#)

Figura 7.5: Ejemplo de resultados de wordreference para definiciones

empiezan vacíos, si no se realizase ninguna acción de ese tipo, al final no se mostrarían en el editor.

De este modo, el texto resultante de hacer una ejecución será todos estos texto puestos uno detrás de otro, siempre en el mismo orden:

1. Texto subrayado
2. Texto del análisis morfológico
3. Texto con los antónimos
4. Texto con los sinónimos
5. Texto con las definiciones
6. Texto con las traducciones

Si se ejecuta alguna acción que necesite acceder a la web de WordReference, citaremos su web al final del texto para que el usuario pueda saber de donde se ha sacado esa información, y debido a los derechos de autor que tiene la web de WordReference. Concretamente, WordReference deja citar pequeñas entradas del diccionario, siempre y cuando se cree un vínculo a su página web.

prueba  ESCUCHAR: ESPAÑA

[Definición](#) | [Sinónimos](#) | [Conjugator](#) | [in context](#) | [images](#)

Inflexiones de 'prueba' (n): [fpl](#): pruebas

Del verbo **probar**: ([conjugar](#))
prueba es:
 3ª persona singular (él/ella/usted) presente indicativo
 2ª persona singular (tú) imperativo

WordReference [Collins](#)

WordReference English-Spanish Dictionary © 2015:

Principal Translations		
prueba <i>nf</i>	(evidencia del delito)	evidence <i>n</i>
	La policía científica está analizando la prueba que puede ser decisiva.	
prueba <i>nf</i>	(comprobación de la división)	check, test <i>n</i>
	Este mes, los alumnos aprenderán a hacer la división y la prueba.	

Figura 7.6: Ejemplo de resultados de WordReference para traducciones

Finalmente, el texto resultante de realizar la ejecución, se le devolverá como respuesta a la parte web de la aplicación para que pueda mostrar el resultado de enriquecer el texto al usuario.

Capítulo 8

Evaluaciones heurísticas

Como ya hemos adelantado en el capítulo 4.2, al final de todo el diseño guiado por objetivos realizamos evaluaciones heurísticas con Guillermo Jiménez y Pablo Moreno, profesores de la Universidad Complutense de Madrid, expertos en usabilidad.

Lo primero que se hizo fue describir a los evaluadores como funcionaba el sistema. Una vez hecho esto, cada evaluador redactó un informe con los problemas encontrados y las heurísticas violadas.

Una vez que teníamos estos informes, estuvimos estudiándolos e hicimos una lista de tareas que arreglarían todos los problemas más importantes. Para determinar qué problemas eran más importantes nos basamos mucho en los niveles de severidad que nos puso Pablo Moreno, y en los que vimos que Guillermo Jiménez daba mayor énfasis en el informe.

La lista de tareas que realizamos para arreglar estos problemas fue la siguiente:

1. Solucionar bug de filas cambiantes: en cierto punto del sistema, la aplicación cambiaba la columna de las acciones por el de las piezas.
2. Reemplazar “Restaurar” por sobrescribir ejecución: nos dimos cuenta que el botón de Restaurar no era muy intuitivo a la hora de deshacer ejecuciones, y decidimos que era mejor eliminar la función de restaurar y simplemente lanzar una nueva ejecución sobre el texto original sin enriquecido, cada vez que se aplicara el enriquecimiento.
3. Feedback por tiempo de procesamiento largo: vimos que no poníamos suficiente feedback cuando se lanzaba una ejecución, y el usuario se sentía perdido y frustrado cada vez que le daba al botón de aplicar. Decidimos arreglarlo poniendo un pop-up que informase al usuario de que su texto se estaba procesando.

4. Signos de puntuación no son útiles: según los resultados de la evaluación, la pieza de signos de puntuación no tenía ninguna utilidad para el usuario, y encima provocaba varios errores graves en el sistema. Por estas razones decidimos eliminar esta pieza.
5. Piezas “P.Difíciles” y “Sig.Puntuación” poco intuitivas: en las evaluaciones descubrimos que el nombre de estas piezas no eran suficientemente descriptivas y decidimos poner “Palabras difíciles” y eliminar los signos de puntuación por no tener utilidad para el sistema.
6. Confusión con las pestañas de piezas y acciones: al parecer, había cierta confusión por como estaban dispuestas las pestañas de piezas y acciones, que hacían que se confundiesen entre ellas. Decidimos marcar más la transición entre una pestaña y otra, poniendo la interfaz como se encuentra actualmente.
7. Pasar todas las filas directamente: esto no fue un fallo en sí, si no una forma de solucionar el fallo de “Restaurar”. Antes pasábamos una a una todas las filas, debido a la antigua funcionalidad del sistema, pero al cambiar la funcionalidad de Restaurar, decidimos pasar todas las filas de la ejecución en una sola llamada al servicio.

A estas tareas les asignamos una puntuación para saber la relevancia que tenían los fallos y saber en qué orden actuar. Las puntuaciones iban del uno al cinco, siendo el 1 lo más prioritario. Estas puntuaciones se recogen en la Tabla 8.1:

Nº de Tarea	Puntuación	Razón de la puntuación
1	1	Bajo coste del arreglo y mucha importancia
2	1	Máxima prioridad
3	5	Muy poco prioritario
4	1	Arregla muchos fallos
5	2	Fácil de arreglar
6	2	Estético y funcional
7	1	Máxima prioridad

Tabla 8.1: Puntuaciones asignadas a las tareas.

Finalmente, hicimos la última iteración de refinamiento y desarrollo del diseño guiado por objetivos en la que arreglamos los problemas de usabilidad más importantes realizando estas tareas por orden de prioridad, realizando primero las tareas de prioridad uno, luego las de dos, y luego la de cinco.

Esto dió como resultado la aplicación que tenemos actualmente.

Capítulo 9

Conclusiones y trabajo futuro

9.1. Conclusiones

Para concluir este trabajo, en este apartado se cuenta cómo se han cubierto las motivaciones iniciales del proyecto y las conclusiones a las que se han llegado.

Para empezar, existen varios sistemas cuyo objetivo consiste en facilitar la comprensión de textos escritos. Se ha investigado sobre tres sistemas como son PorSimples, *Open Book* y NavegaFácil que llevan a cabo estos objetivos de distintas formas, para hacer un análisis de competencias. *Open Book* realiza esta labor sobre textos que el usuario desea comprender, y PorSimples y NavegaFácil sobre contenido en páginas web.

En cuanto a las tecnologías que hemos usado en la aplicación, hemos decidido usar tecnologías web, ya que favorecen la distribución a cualquier plataforma. Concretamente, hemos usado Bootstrap con Material design para que la interfaz tenga un diseño actual con el que el usuario se sienta cómodo. Además, se ha implementado un servicio web RESTful, ya que utiliza el protocolo HTTP que es el método de comunicación más utilizado actualmente y ayuda a que cualquier persona pueda utilizarlo. También estudiamos la posibilidad de incluir otras tecnologías como servicios web SOAP o WSDL, pero al final decidimos usar la arquitectura REST dado que se adaptaba mejor a las necesidades del proyecto.

Para que la aplicación fuese más ligera y para favorecer su uso en muchas plataformas, se ha usado una arquitectura cliente-servidor para que toda la carga del procesamiento del texto recaiga en el servidor y no en la aplicación. Para ello se ha desplegado el servicio web RESTful mencionado anteriormente en un servidor de la universidad, para que cualquiera pueda acceder a él, y se ha implementado el cliente en la parte de la aplicación.

Además, en cuanto a la aplicación, se ha dado mucha importancia a la usabilidad de interfaces. Se ha seguido una adaptación del Diseño Guiado por

Objetivos para nuestro proyecto que ha ayudado a encontrar muchos errores de usabilidad y hacer la aplicación más amigable para el usuario. Para ello se han realizado evaluaciones con usuarios y expertos en usabilidad.

También es muy importante que esta aplicación, cuyo propósito es ayudar a la gente, pueda ser modificada por cualquier persona. Para ello, se ha publicado todo el código y la memoria en Github con licencia libre para que cualquiera pueda modificar y mejorar el programa. De esta manera se puede conseguir ayudar cada vez a más gente con las mejoras que se hagan entre todos. Se puede encontrar en la página <https://github.com/jaimedelgado/Xtext>.

9.2. Trabajo futuro

En vista del resultado obtenido al finalizar el desarrollo del proyecto, cabe replantearse la posibilidad de futuras iteraciones que añadan nuevas características al sistema. Debido al tiempo limitado del proyecto, no se han podido implementar todas las ideas que surgieron a lo largo del desarrollo. En el caso de querer retomar este proyecto para continuar con su desarrollo, sugerimos todas estas ideas que surgieron como posibles modificaciones del sistema.

En primer lugar, habría que sustituir el editor CKEditor por la versión que están desarrollando para dispositivos móviles, ya que con esta versión el editor puede ser visualizado en cualquier tamaño de pantalla.

También, se podrían añadir funciones para exportar e importar el texto enriquecido junto con las piezas y acciones que se aplicaron para que el usuario pueda volver a modificarlo en un futuro o pueda distribuirlo como desee. También sería interesante añadir funciones para guardar y cargar texto en otros formatos para que el usuario pueda enriquecerlos.

Además, se podría facilitar al usuario el acceso a los textos enriquecidos con anterioridad mediante la incorporación de un historial en la interfaz que muestre dichos textos junto con su hora de la última modificación.

Sería interesante incorporar un gestor de piezas y acciones que posibilite al usuario la creación, eliminación o incluso la edición de piezas y acciones en el sistema.

También se podrían añadir nuevas acciones por defecto al sistema para facilitar aún más la comprensión del texto como por ejemplo obtener imágenes o información de la Wikipedia.

Además, sería interesante permitir al usuario la posibilidad de crear una acción como resultado de la combinación de una o varias filas de piezas y acciones para su posterior uso.

Por otro lado, el sistema guarda para una sesión toda la información sobre los textos que los usuarios enriquecen así como las piezas y acciones que aplican para tal fin. Si esta información se guardase también en una base de datos se podrían obtener estadísticas sobre qué tipo de textos se suelen enriquecer así como qué tipo de textos tiene mayor complejidad para su comprensión. En el caso de las piezas y acciones también se podrían obtener estadísticas sobre cuáles de ellas son más utilizadas por los usuarios y así saber cuáles son los problemas más frecuentes a la hora de comprender texto escrito.

Por último, como todos los dispositivos móviles actuales disponen de GPS, sería más que interesante obtener la geolocalización de los dispositivos desde los que los usuarios utilizan el sistema para poder obtener estadísticas sobre qué regiones tienen mayor dificultad a la hora de comprender texto escrito y en qué consiste dicho problema.

Capítulo 10

Conclusions and future work

10.1. Conclusions

To conclude this work, this section explain how we have covered the initial motivations of the project and the conclusions that have been reached.

First of all, there are several systems which aims to facilitate the understanding of written texts. Was investigated on three systems such as PorSimples, Open Book and *NavegaFácil* that carrying out these objectives in different ways, to make an analysis of skills. Open Book does this work on texts which the user wants to understand, and PorSimples and *NavegaFácil* on content web pages.

As for the technologies we have used in the application, we decided to use web technologies because favore their distribution on any platform. Specifically, we have used Bootstrap with Material design for the interface has a modern design with which the user feel comfortable. In addition, we have implemented a RESTful web service, since it uses the HTTP protocol that is the communication method most commonly used type and helps anyone can to use it. We also study the possibility of including other technologies such as SOAP or WSDL web services, but eventually decided to use the REST architecture since adapted to the needs of the project.

For the application was lighter and to encourage its use in many platforms, we have used a client-server architecture for all text processing load is put on the server and not on the application. For this we have deployed the RESTful web services mentioned above on a university server, so anyone can access it, and has been implemented on the client side of the application.

Furthermore, with regard to the application, it has given much importance to the usability of interfaces. We have followed an adaptation of the Goal-Centered Design by our project that has helped find many mistakes usability and make the application more user-friendly. For that we have realized evaluations with users and usability experts.

It is also very important for this application, which aims to help people, it can be modified by anyone. To this end, we have published all the code and memory in Github freely licensed for anyone can modify and improve the program. On this way the application can get help more and more people with the improvements to be made together. It can be found on page <https://github.com/jaimedelgado/Xtext>.

10.2. Future work

In view of the result obtained at the end of the project, it should reconsider the possibility of future iterations that add new features to the system. Due to the limited time of the project, we have not been able to implement all the ideas that emerged during the development. In case it wants to take up this project for further development, we suggest all these ideas that emerged as potential system modifications.

First, we should replace the editor CKEditor version being developed for mobile devices and with this version the editor can be displayed on any screen size.

Also, add functions to export and import the enriched text with the pieces and actions that were applied so that the user can re-edit it in the future or can distribute as he wish. Also, add functions to save and load text into other formats so that users can enrich them.

In addition, facilitate user the access to text previously enriched by incorporating a history in the interface that showing these texts along with their time of last modification.

It would be interesting to incorporate a pieces and actions manager that enables the user to create, delete or edit pieces and actions in the system.

It would also be interesting to allow the user to create an action as a result of the combination of one or more rows of pieces and actions for later use.

In addition, add new actions to the system to further facilitate understanding of the text such as obtain images or information from Wikipedia.

On the other side, the system saves for a session all information about the texts that users enrich as well as pieces and actions that apply for that purpose. If this information can also keep it in a database we could get statistics on what type of texts are enriched as well as what type of texts has more complexity for their understanding. In the case of the pieces and actions could also obtain statistics about which ones are most used by users and then we could know what the most common problems to understand written text.

Finally, like all current mobile devices have GPS, it would be interesting to get the geolocation of the devices from which users use the system to get statistics about which regions of the world have more difficulty understanding written text and what is the problem.

Capítulo 11

Trabajo individual

En este capítulo se expone la aportación de cada miembro del grupo al proyecto.

11.1. Jaime Delgado Linares

Al comenzar el proyecto, nuestras tutoras nos dieron bastante documentación acerca del procesamiento de lenguaje natural, que incluían información acerca del análisis sintáctico y léxico de texto, así como información de la herramienta Mate. Mi trabajo al principio del proyecto fue leerme toda esta documentación y aprender todo lo referente al procesamiento de lenguaje natural. Como no había trabajado mucho ese campo tuve que buscar bastante información acerca de esto, ya que era importante tener una base sólida para el resto de proyecto. Además, nos dieron acceso a la máquina que íbamos a usar como servidor, y mi trabajo también fue la de familiarizarme con el servidor, e instalar todo lo necesario.

Dentro de esta fase preparativa del proyecto, también tuve que aprender a usar \LaTeX para hacer la memoria. Decidimos usar \LaTeX sobre todo porque queríamos aprender algo nuevo, y porque nuestras tutoras nos lo recomendaron. Mi trabajo en ese momento también fue la de buscar información acerca de \LaTeX , y sobre todo, aprender como se usaba la plantilla de \TeX IS. También hice plantillas para ayudar a mi compañero a aprender a usar \LaTeX , y preparé el proyecto en Bitbucket para que pudieramos gestionar ahí la memoria.

Una vez que ya teníamos todo preparado, empezamos a hacer una especificación del sistema en la que decidimos las funciones que iba a comenzar teniendo el sistema, y la arquitectura que iba a tener. Decidimos que iba a ser un cliente-servidor para rebajar la carga dentro del dispositivo en el que se encontrase. De este modo, el siguiente paso fue investigar acerca de los distintos tipos de servicios web.

Mientras que mi compañero investigaba acerca de servicios web SOAP, REST y sobre Android, yo me centré en investigar acerca de servicios web WSDL y de JAX-WS, ya que los servicios web de IULA usaban este protocolo y esa herramienta. También estuve investigando como usar estos servicios web de IULA en nuestro proyecto, y qué es lo que hacían estos servicios. Esto significaba estudiar todo lo referente a MaltParser y Freeling, así como decidir cuáles de estos servicios web íbamos a usar en nuestro proyecto.

El siguiente paso, una vez que ya sabía acerca de Freeling, MaltParser, y servicios web, fue hacer una pequeña aplicación cliente en java que usase estos servicios. Cuando conseguí usar estos servicios, mi trabajo fue parsear las salidas resultantes para realizar las primeras funcionalidades del sistema y todas las piezas menos las palabras difíciles. Primero empezamos por subrayar tokens, ya que era la función más fácil de realizar con estos servicios, y después, empezamos la comunicación con la parte web.

Entre mi compañero y yo, decidimos convertir mi aplicación inicial en un servicio web RESTful. Esto funcionaba con textos sin tildes ni caracteres especiales. Mi trabajo aquí fue investigar por qué el sistema no aceptaba estos caracteres, y entre los dos, descubrimos que era porque usabamos QueryParams y los caracteres especiales eran caracteres HTML que empezaban todos por &. Esto hizo que tuvieramos que investigar mucho y decidiesemos usar JSON.

Con la nueva estructura usando JSON, y con la incorporación de nuevas funcionalidades como el análisis morfológico, y a punto de incorporar el resto de funcionalidades, decidí cambiar toda la estructura interna de la aplicación para que fuese más fácil de reutilizar. Esto lo decidí debido a que el sistema, hasta ese momento, sólo estaba pensado para que funcionara con lo que teníamos, y creí adecuado modificarla para que fuese más fácil añadir el resto de funcionalidades.

Una vez que hice la nueva estructura, creé la pieza “palabras difíciles” e hice el resto de funcionalidades usando la página web de WordReference. Aquí tuvimos un par de problemas de comunicación con la página que me ayudó a resolver mi compañero.

Después de tener todas las funcionalidades, ya sólo quedaba ir haciendo pruebas del sistema e ir arreglando en varias iteraciones todos los errores que veíamos. Mientras hacía esto, también estuve escribiendo la memoria del proyecto, concretamente he redactado e ilustrado los siguientes apartados:

- Los resúmenes en español e inglés.
- El apartado “1.1. Motivación” del capítulo “1. Introducción”.
- El capítulo “2. Introduction”.

- Los apartados de “3.1. Procesamiento del Lenguaje Natural (PLN)” y “3.2.5. JAX-WS” del capítulo “3. Estado del Arte”.
- Los apartados “4.1. Diseño Guiado por Objetivos (DGO)” y “4.2. Adaptación del Diseño Guiado por Objetivos para el proyecto” del capítulo “4. Diseño y requisitos”.
- Parte del apartado “5.2. Servicio web” del capítulo “5. Arquitectura”.
- El capítulo “7. Servidor”.
- El apartado “9.1. Conclusiones” del capítulo “9. Conclusiones y trabajo futuro”.
- El capítulo “10. Conclusions and future work”.

11.2. Juan Luis García Flores

Para empezar, tuve que investigar sobre las distintas tecnologías existentes para saber cómo combinarlas y pensar una estructura adecuada y eficiente del sistema. En las primeras reuniones que hicimos con nuestras tutoras decidimos hacer una estructura cliente servidor ya que así el cliente no tendría que realizar todo el procesamiento requerido para enriquecer texto dado que el servidor tendría un servicio web que lo procesaría.

Pensado esto, comencé investigando sobre servicios web SOAP, Android y como establecer la comunicación entre ellos. Construí dos pequeños sistemas para poner en prueba lo investigado, una aplicación implementada en Android que se comunicaba mediante SOAP con un servicio web construido con ASP y la misma aplicación Android que se comunicaba mediante SOAP con un servicio web construido en Java.

Tras comentarnos nuestras tutoras sobre los servicios web REST y la API Jersey de Java, decidí investigar sobre esto y construí un servicio web que comunicase con la aplicación en Android que ya estaba implementada.

Una vez logrado esto, empecé a investigar sobre las tecnologías web existentes para sustituir la aplicación en Android por una web. Esta investigación surgió porque en una asignatura llamada “Interfaces de usuario” empezábamos a estudiar HTML5. Tras ver en la asignatura herramientas como Bootstrap y Bootstrap con Material design que cuentan con diseño *responsive* y haber realizado una práctica con tres compañeros, decidí investigar sobre cómo convertir una web en una aplicación de Android nativa así como la comunicación entre el servicio web REST y la web con la herramienta AJAX. Esta idea era más viable dado que era mucho más fácil enriquecer texto con las propias etiquetas de HTML5 y se podría entonces construir una interfaz multiplataforma.

Tras la investigación construí una web muy básica que comunicaba con el servicio web REST que ya había implementado y lo subí al servidor que nos proporcionaron nuestras tutoras para poder comprobar la comunicación.

Vista que esta idea era mucho más viable, contruí el boceto con todas las pantallas para realizar las dos evaluaciones que me encargué de preparar con nuestras tutoras.

Dediqué la mayor parte del tiempo del desarrollo del proyecto a la construcción por completo de la interfaz propuesta en el boceto. Para su construcción tuve que investigar sobre el editor CKEditor y la forma de incluirlo en la interfaz, así como el lenguaje JavaScript para implementar la lógica de la interfaz, y JSON para enviar los datos al servicio web con un formato adecuado.

En cuanto al servicio web, ayudé a establecer la comunicación con la web de WordReference para poder implementar las acciones de Sinónimos, Antónimos, Definición y Traducción a inglés.

Debo añadir que todo lo que tuve que investigar, tanto lenguajes nuevos como todo lo relativo a servicios web y la comunicación con éste era nuevo para mí (salvo HTML y Bootstrap que se introdujo en la asignatura de Interfaces de usuario) y tuve que estudiar sobre ello.

Para concluir, respecto a la memoria he redactado e ilustrado con imágenes y ejemplos:

- Los apartados “Objetivos” y “Estructura del documento” de la introducción.
- El capítulo de “Estado del arte” salvo el apartado de “Procesamiento del Lenguaje Natural”.
- El apartado “Desarrollo del diseño en el proyecto” del capítulo “Diseño y requisitos”.
- El capítulo de “Arquitectura” prácticamente por completo.
- El capítulo de “Aplicación de enriquecimiento”.
- El apartado de “Llamada al servicio principal” del capítulo "Servidor".
- Y el apartado de “Trabajo futuro” del capítulo “Conclusiones y trabajo futuro”.

Bibliografía

- ALBERTO FERNÁNDEZ. Servicios web RESTful con HTTP. 2013. Disponible en <http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas> (último acceso, November, 2013).
- ALEX NGHIEM. The Basic Web Services Stack. Disponible en <http://www.informit.com/articles/article.aspx?p=31076&seqNum=2>.
- ALUÍSIO, S. M., SPECIA, L., PARDO, T. A., MAZIERO, E. G. y FORTES, R. P. Towards Brazilian Portuguese Automatic Text Simplification Systems. En *Proceedings of the eighth ACM symposium on Document engineering*, DocEng '08, páginas 240–248. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-081-4.
- BAGÓ, A. y GARCÍA, J. Navegafácil. Disponible en <http://hypatia.fdi.ucm.es/proyecto/web/navegafacil.php>. Trabajo de Fin de Grado, Facultad de Informática, UCM.
- BARBU, E., MARTÍN-VALDIVIA, M. T. y UREÑA LÓPEZ, L. A. Open Book: a tool for helping ASD users' semantic comprehension. En *Proceedings of the Workshop on Natural Language Processing for Improving Textual Accessibility (NLP4ITA)*. 2013.
- BERNERS-LEE, T. y JAFFE, D. J. World Wide Web Consortium. 1994. Disponible en <http://www.w3c.es/> (último acceso, March, 2015).
- CHAO, I., CARRERAS, X., PADRÓ, M. y PADRÓ, L. *Freeling: An Open-Source Suite of Language Analyzers*. Proceedings of the Fourth International Conference on Language Resources and Evaluation, 2004.
- CKSOURCE. The best web text editor for everyone. Disponible en <http://ckeditor.com/>.
- ECMA INTERNATIONAL. Introducing JSON. Disponible en <http://json.org/>.
- GLASSFISH COMMUNITY. JAX-WS. Disponible en <https://jax-ws.java.net/>.

- GOOGLE. Material design. Disponible en <http://www.google.com/design/>.
- HALL, J., NILSSON, J. y NIVRE, J. Maltparser. 2007. Disponible en <http://www.maltparser.org/> (último acceso, November, 2014).
- JIMÉNEZ, G., MORENO, P. y SÁNCHEZ, A. *Tema 4 Desarrollo de Sistemas Interactivos*. Facultad de Informática, UCM, 2015.
- KELLOGG, M. WordReference. Disponible en <http://www.wordreference.com/es/>.
- ORACLE. JAX-RS: Java API for RESTful WebServices. Disponible en <https://jcp.org/en/jsr/detail?id=311>.
- TWITTER. Bootstrap. Disponible en <http://getbootstrap.com/>.
- UNIVERSIDAD DE VIGO. Sistemas Cliente/Servidor. 2008. Disponible en <http://ccia.ei.uvigo.es/docencia/SCS/0910/transparencias/Tema4.pdf> (último acceso, October, 2008).
- W3SCHOOLS. Ajax. Disponible en <http://www.w3schools.com/ajax/>.
- W3SCHOOLS. CSS3. Disponible en http://www.w3schools.com/css/css3_intro.asp.
- W3SCHOOLS. HTML5. Disponible en http://www.w3schools.com/html/html5_intro.asp.
- W3SCHOOLS. JavaScript. Disponible en <http://www.w3schools.com/js/default.asp>.
- W3SCHOOLS. JQuery. Disponible en <http://www.w3schools.com/jquery/default.asp>.
- W3SCHOOLS. JQuery Mobile. Disponible en <http://www.w3schools.com/jquerymobile/>.
- WIKIPEDIA (Representational State Transfer). Entrada: "Representational State Transfer". Disponible en http://es.wikipedia.org/wiki/Representational_State_Transfer (último acceso, May, 2015).